Escuela Politécnica Superior

Roboto Runner: Desarrollo y publicación de un videojuego para dispositivos móviles.

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Manuel Font Rodríguez

Tutor/es:
Carlos José Villagra Arnedo

Universitat d'Alacant
Universidad de Alicante

JULIO 2021

Resumen

La llegada de los dispositivos móviles inteligentes ha generado un gran marco de desarrollo para los videojuegos. Al mismo tiempo que esta serie de teléfonos, tabletas o incluso pulseras, se iban afianzando en el uso cotidiano, los videojuegos no pararon de evolucionar y de intentar afianzarse en este sector.

Y vaya si lo consiguieron, ha habido juegos como el famoso *Flappy Bird*, que han conseguido tener millones y millones de usuarios diarios. El nicho de mercado fue creciendo desde su inicio e incluso actualmente no se ha parado en ningún momento el crecimiento exponencial de este.

Los videojuegos, aunque existían antes de que los dispositivos móviles inteligentes vieran la luz, no habían tenido una concepción como la requerida en este tipo de sistemas. Con la expansión de estos dispositivos se cambiarían las reglas, se comenzaría a premiar los juegos con un estilo Arcade parecido al ya visto con las famosas recreativas. Del mismo modo, la gran mayoría de ingresos llegarían a través de la publicidad y no por el precio del juego.

En base a esta información, nace la idea del proyecto en el que está basado este proyecto, Roboto Runner. Este es un videojuego de estilo 2D y basado en el género ¨Endless Runner¨, el cual está caracterizado por priorizar la diversión a cualquier otro apartado. Ha sido desarrollado en Unity y publicado en la plataforma Play Store de Google.

Con todo esto y si tuviéramos que reducir los contenidos a mostrar en un párrafo, podríamos decir que: se va a ver el diseño, el desarrollo y la implementación de un videojuego para dispositivos Android. Asimismo, se van a explicar distintos apartados para poder entender mejor la viabilidad del proyecto y su correspondiente estudio de mercado, donde además se va a dar la posibilidad de llegar a comprender la competencia del proyecto y visualizar el gran público objetivo al que aspira el videojuego. Finalmente, se va a observar cómo ha sido la publicación del juego y los resultados que esta conlleva.

Dedicatoria

Acabar este proyecto supone el fin de una etapa y el comienzo de otra. Y como todo cambio, conlleva su alegría, su satisfacción y como no, su melancolía.

Hay tantas experiencias vividas a lo largo de estos años que no sé cómo hacer que salgan de mi mente de una forma ordenada.

Para empezar, me gustaría agradecer a todos y cada uno de los profesores que me han ayudado a lo largo de este proceso y, siento no haber sido el mejor alumno en algunas ocasiones.

Del mismo modo, gracias a todos y cada uno de mis compañeros, a los que siguen en contacto y también, a los que no. Tengo la suerte de poder decir que he conocido mucha gente maravillosa. Y en especial, gracias a ti Adrián, fuiste mi hermano mayor durante gran parte de esta etapa y al menos para mí, siempre lo serás.

También me encantaría recordar a todos mis compañeros de Omu, con vosotros pasé momentos geniales e hicimos un trabajo que será difícil de olvidar.

Asimismo, no puedo olvidarme de mi tutor en este proyecto, muchísimas gracias Carlos, has sido mejor tutor de lo que he merecido y eres uno de los mayores profesionales que he conocido. Eternamente agradecido.

De la misma manera, no puedo estar más feliz de poder agradecerle a mis abuelos todo lo que me han dado a lo largo de los años. A ti yeye, mi mejor amigo desde los tres años y a ti yaya, la persona más importante de mi vida. No puedo imaginar una vida sin vosotros.

Igualmente, quiero agradecer a mi hermano, la situación actual es difícil pero algún día la tempestad cederá y volveremos a ser lo que fuimos. A mi tío José, el tiempo pasa pero nuestra relación no, ojala tuviéramos más tiempo para poder pasarlo junto. A mi padre putativo, posiblemente la mejor persona que conozco y seguro, la persona con más paciencia del mundo. Estés aquí o allá, de mí no te librarás.

Quiero agradecerte a ti, Julia, apareciste en la penumbra e iluminaste con tu luz la senda. No sabes cuánto te agradezco todo el proceso que me ayudaste a pasar. Ojalá podamos compartir la vida juntos.

Y me gustaría finalizar con el agradecimiento a mis padres. A mi madre, la persona que me ha cuidado siempre y a la que nunca podré agradecérselo suficiente. Mil y una gracias por estar a mi lado siempre, sin ti nada sería posible. Y a mi padre, empezamos este camino juntos y lo acabaré contigo, aunque sea en mi recuerdo. Espero que estés orgulloso de mí y puedas seguir los pasos de tus hijos allá dónde estés.

Contenido

Glosario de términos

Android
Android es el nombre de un sistema operativo que se emplea en dispositivos móviles, por lo general con pantalla táctil. 24, 39, 58, 59, 71, 73, 83, 89
Canvas
El Canvas es el área donde todos los elementos de la intefaz de usuario deben estar. 97
constructor
En Programación Orientada a Objetos (POO), un constructor es una subrutina cuya misión es inicializar un objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto.....105
coste computacional
Coste que tiene un sistema de información a nivel de hardware, software y mantenimientos. 99
diagrama de flujo
El diagrama de flujo o flujograma o diagrama de actividades es la representación gráfica de un algoritmo o proceso.....93
GDD
Acrónimo del inglés Game Design Document (Documento de Diseño de Juego). 88, 95
Google Meet
Google Meet es la aplicación de videoconferencias de Google, para navegadores web y dispositivos móviles, enfocada al entorno laboral. 59
hardware
Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático. 41, 43, 44, 46, 47, 54, 56, 57, 58, 59
hitbox
Una hitbox es una técnica invisible comúnmente utilizada en los videojuegos para la detección de colisiones en tiempo real, también llamadas cajas delimitadoras o cajas de colisión......99
HUD
Información relativa al estado de la partida y que permanece visible en pantalla durante la partida. 37
impresiones
las impresiones son la métrica que refleja el número de veces que se muestra un contenido. 56, 63, 64, 65, 113
inspector de Unity
El Inspector es usado para ver y editar propiedades de un objeto y también preferencias y otros ajustes dentro de Unity. 92, 95
juegos AAA
Un juego AAA (comúnmente llamado Triple A) es una clasificación informal utilizada para los videojuegos producidos y distribuidos por una distribuidora importante o editor importante. 88
lore
Es el conjunto de historias, datos, personajes, representaciones, etc que conforman el universo representado y le dan coherencia 22, 73
memoria RAM
La memoria RAM es la memoria principal de un dispositivo, esa donde se almacenan de forma temporal los datos de los programas que estás utilizando en este momento. 101
parallax
Efecto que consiste en simular el desplazamiento de los objetos a diferentes velocidades y en distintas capas. 37, 68, 77, 84, 96

Índice de ilustraciones

Índice de tablas

1 Introducción

El impacto de los dispositivos móviles en la rutina cotidiana de una persona ha sido inaudito. Redes sociales, navegadores móviles, aplicaciones destinadas a intentar controlar el sueño, a mejorar rutinas de entrenamiento y un largo etcétera. Miles y miles de aplicaciones distintas tratan de convivir día a día con los humanos debido a la expansión de teléfonos, pulseras y una serie de dispositivos electrónicos inteligentes.

Pero no todas las aplicaciones han de estar relacionadas con facilitar la vida de las personas, hay otro gran sector encargado del disfrute y la utilización del tiempo libre de estas mismas personas. Dentro de este nicho de ocio, podríamos encontrar los videojuegos.

Estos, existían previamente de que los dispositivos móviles inteligentes vieran la luz. Sin embargo, su constante expansión fue creando un nicho de mercado cada vez más grande. Un nicho de posibles clientes caracterizados por la búsqueda de experiencias de juego rápidas, directas, sin apenas explicaciones ni introducciones, pero al unísono, divertidas y entretenidas.

Y ha de ser en este nicho de mercado, donde Roboto Runner (nombre del videojuego en cuestión) haga un esfuerzo para intentar hallar un pequeño hueco donde expandirse.

Este documento ha sido organizado de tal forma que contenga todos los apartados necesarios para maximizar el entendimiento del proyecto en cuestión. En coherencia con esta información, se va a hacer una pequeña introducción a cada uno de los apartados principales.

En primer lugar, en este mismo capítulo, la Introducción, se está haciendo un pequeño recorrido por todos los apartados que conforman o componen la memoria del proyecto.

A continuación, se podrá ver el Marco Teórico, donde se hará una definición del estilo y del género elegidos para la realización del videojuego. Asimismo, se podrá ver varios referentes importantes del género y sus características principales. Toda esta información ayudará a contextualizar Roboto Runner.

Seguidamente, se procederá a explicar los Objetivos planteados para la realización del proyecto. Donde se podrá ver como el objetivo principal se desglosará en objetivos más concretos para facilitar el desarrollo del videojuego.

En el apartado posterior, Metodología, se podrá conocer el método seleccionado para el desarrollo y el porqué de esta elección. Del mismo modo, se podrá ver también, cuáles han sido las técnicas elegidas para la gestión y planificación del proyecto.

Posteriormente, en el Análisis, se generará un profundo estudio sobre ciertos apartados del proyecto. Se identificarán tanto los requisitos funcionales como los no funcionales previstos. Una vez esta información haya tomado forma, se usará para componer un estudio de viabilidad que dictaminará la estabilidad del proyecto. Este estudio estará dividido en: gestión de riesgos (se acotarán las posibles adversidades que podrían surgir y se asignarán distintas estrategias para saber cómo actuar), estimación de costes (se verán todos los componentes de costes que sumados acabarán formando el coste final del proyecto), modelo de negocio (se observarán los métodos seleccionados para intentar generar beneficios) y nicho de mercado (se explicará el público objetivo al que el videojuego aspirar llegar).

Después, en el apartado Diseño del juego, se definirá la ficha técnica que posteriormente compondrá el videojuego. Además de esto, se visualizarán y detallarán gran parte del contenido visual que conforma Robot Runner: historia, estilo de género, ambientación, personaje principal, escenario, enemigos, jugabilidad, mecánicas, controles y escenas.

Más tarde, en el Desarrollo, se intentará explicar cómo se ha procedido durante la realización del videojuego. Este compone el apartado más técnico del documento e incluso se han incluido diagramas y fragmentos de código para facilitar la comprensión del apartado y destacar los apartados más importantes.

Ulteriormente, en los Resultados, se mostrará el resultado final del proyecto consistente en la subida a la plataforma Play Store del videojuego previamente desarrollado. De igual modo, se podrán ver algunas herramientas provistas por Google para hacer un seguimiento de la evolución de una aplicación.

Luego, en las Conclusiones, se expondrán los resultados conseguidos en base a los objetivos previamente descritos y, los conceptos aprendidos durante el transcurso del proyecto.

Para finalizar, en el apartado Bibliografía, se podrán consultar las fuentes utilizadas para la extracción de la información descrita a lo largo del documento.

2 Marco Teórico

En este apartado se procederá a explicar los principios de este tipo de videojuegos, tanto a nivel jugable (el apartado principal de este género de videojuegos) como a nivel estético. A su vez, se podrán ver varios ejemplos de videojuegos destacados en este tipo de género (Endless Runner), y en base a estos explicaremos qué parte de esos juegos, han intervenido inspirando, aunque sea levemente en el desarrollo de este proyecto.

Asimismo, haremos un recorrido en que nos hemos inspirado visualmente para desarrollar el contenido visual del proyecto. Es decir, no toda la inspiración viene directamente de videojuegos, también intervienen varios artistas de otras temáticas no directamente involucrados en el mundo del desarrollo de videojuegos.

2.1 Definición videojuego

Un **videojuego** es una aplicación interactiva orientada al entretenimiento que, a través de ciertos mandos o controles, permite simular experiencias en la **pantalla** de un televisor, una computadora u otro dispositivo electrónico.

Los videojuegos se diferencian de otras formas de entretenimiento, como ser las películas, en que deben ser interactivos; en otras palabras, los usuarios deben involucrarse activamente con el contenido. Para ello, es necesario utilizar un mando (también conocido como gamepad o joystick), mediante el cual se envían órdenes al dispositivo principal (un ordenador o una consola especializada) y estas se ven reflejadas en una pantalla con el movimiento y las acciones de los personajes (Gardey, 2013).

2.2 Endless Runner

*Del inglés **endless runner** (corredor infinito).*

Género donde el jugador debe avanzar de manera irremediable en una misma dirección, generalmente escapando de algún enemigo o peligro, y cuyo objetivo es avanzar lo máximo posible antes de morir.

Generalmente las acciones principales son saltar y esquivar obstáculos, pero muchos juegos de este tipo incluyen también algún tipo de ataque, tanto cuerpo a cuerpo como con armas (GamerDic, 2014).

Es un género basado en la jugabilidad, que es el factor clave en el desarrollo de un videojuego de este tipo. Mayoritariamente, son juegos sin historia, sin un “lore” profundo, y en algunos casos siquiera sin un poco del mismo. Es decir, carecen de un pretexto claro, el abanico va

desde un pájaro que tiene que pasar entre tuberías del Mario Bros, hasta una rana que tiene que atravesar una autovía infinita, sorteando automóviles a su paso.

Comúnmente, empieza la partida sin introducción alguna. Una vez le das a empezar la partida, el personaje principal ya se encuentra avanzando de forma infinita, ya sea corriendo, nadando, volando o de cualquier forma en la que se pueda avanzar sin parar.

El desplazamiento puede ser tanto en X como en Y, aunque es más común el desplazamiento horizontal. Las mecánicas son prácticamente las mismas en ambas direcciones, pero en concreto, para los que están basados en el desplazamiento vertical, hay una mecánica añadida que es la de moverse de izquierda a derecha, sumadas a las típicas en ambos de saltar, rodar o hacer un desplazamiento rápido en alguna dirección.

Este tipo de videojuego se ha visto claramente beneficiado por la creación y posterior auge del mercado en dispositivos móviles. En este mercado se favorece a los juegos con grandes mecánicas, que estén pulidas y sobre todo que sean satisfactorias para el jugador. La historia no es un pilar fundamental, y a pesar de que haya juegos donde se introduce cierto arco argumental, no suele ser lo más común. Con todo esto, no hacen falta demasiadas explicaciones del porqué este género ha crecido tanto en la última década.

Sin embargo, el primer juego de este género data de mucho antes que esto. El primero es 'Speed Race'. Un viejo juego de arcade en blanco y negro lanzado en 1974 en Japón, que nos dejaba controlar un carro de carreras mientras esquivamos a otros vehículos. Mientras más puntuación teníamos, más rápido iba el carro y más angosta era la pista, por lo cual era más difícil evitar obstáculos. Cuando llegó a Occidente, de la mano de Midway Games, su nombre pasó a 'Wheels' (Martínez, 2015).

2.3 Principales referentes del género

Este género tiene gran cantidad de juegos que son referentes para el mismo, vamos a ver algunos:

- **FROGGER**

El primer gran exponente del género, desarrollado originalmente como arcade, en 1981. Fue desarrollado por Konami, y distruibuido por Sega. Frogger es uno de los videojuegos más famosos dentro de la historia de estos mismos (Museum of the Game, 2021). De hecho, casi 40 años después de su lanzamiento, seguimos viendo incontables secuelas y adaptaciones, tanto directamente asociadas, como indirectamente. E incluso fue tal su fama, que a principios de los 80, tuvo una adaptación a la televisión. Desgraciadamente, no duró en antena más que una temporada (mobygames, 2021).

No es un *endless runner* en si mismo, ya que hay un nivel principal con distintos subniveles, es decir, no es un mundo de generación infinita. Sin embargo, sí que es un solo mundo donde la única finalidad es avanzar saltando. El objetivo final es llevar a cada rana a su hogar, pasando

entre coches, ríos y algún que otro obstáculo más. El jugador tiene la capacidad de mover la rana en todas direcciones a través del joystick, aunque el desplazamiento continúo es vertical.

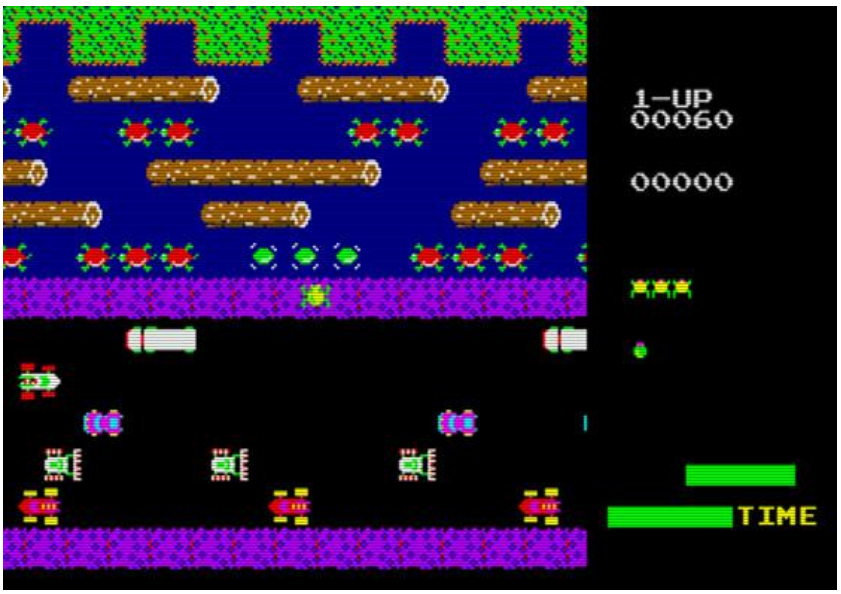

Ilustración 1 Frogger, fuente: https://elblogdemanu.com/frogger/

Actualmente, existe un juego llamado Crossy Road con cierta importancia en el sector, que parte de todas las bases puestas por Frogger en 1981. Éste está desarrollado para dispositivos móviles, disponible tanto en iOS como Android.

Su jugabilidad es muy parecida, enfocada en un desplazamiento en vertical, en el que vamos pasando entre coches y ríos, con la principal diferencia de que aquí sí que tenemos un mundo de generación infinita.

Ilustración 3 Portada videojuego Crossy Road, fuente: Hipster Whale

Ilustración 2 Captura videojuego Crossy Road, fuente: Hipster Whale

- **FLAPPY BIRD**

El más conocido en la actualidad. Fue un juego desarrollado para dispositivos móviles por el desarrollador vietnamita Nguyen Hà Đông, y posteriormente publicado por .Gears Studios (un pequeño estudio de juegos indie). Este juego, publicado el 24 de mayo de 2013, tuvo tal impacto que, en apenas meses, llegó a ser el más descargado en todas las tiendas de aplicaciones móviles. Fue de tal repercusión, que meses antes de cumplir el año después del lanzamiento, el creador comunicó que el juego sería eliminado de todas las plataformas en las que el juego había sido distribuido (Nguyen, 2014).

La prensa inicialmente atribuyó esta desaparición a la enorme cantidad de ingresos que recibía exclusivamente del pequeño banner de publicidad que poseía Flappy Bird el menú (se especula que llegaron a ser hasta 90.000 dólares estadounidenses al día). Más tarde, fue el mismo creador en una entrevista para la revista Rolling Stone, donde dijo que eliminó el juego debido a: *«Había recibido e-mails de fanáticos de Flappy Bird alegando que era tan adictivo como el crack y esto frustraba a la gente»*
Agregó también que, muy por el contrario, su intención inicial era
«crear un juego que las personas pudieran disfrutar durante diez minutos, en un contexto de relax, no dedicarle horas tratando de alcanzar un puntaje superior» (Blázquez, 2014).

Tras la eliminación de Flappy Bird, y de una forma similar a la acontecida con Frogger, son incontables los innumerables juegos que vieron la luz intentando copiar directamente, o al menos inspirándose, en este gran fenómeno.

Ilustración 4 Flappy Bird, fuente: .Gears Studio

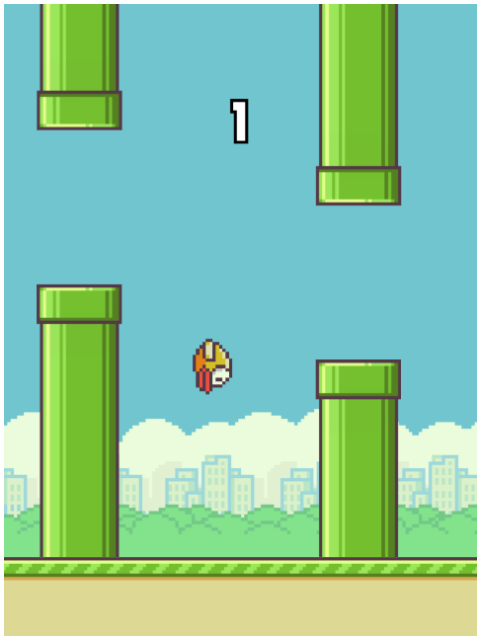

Ilustración 5 Réplica Flappy Bird, fuente: https://flappybird.io/

En cuanto a jugabilidad, el jugador controla al pájaro intentando esquivar esas tuberías verdes tan parecidas a las de un juego de Nintendo. La única mecánica es tocar la pantalla, lo cual generará una pequeña alza en el vuelo del pájaro. El desplazamiento es horizontal, y la generación de mundo es infinita.

- **TEMPLE RUN Y TEMPLE RUN 2**

Ambos desarrollados y publicados por Imangi Studios, y distribuidos a través de todas las plataformas móviles por el mismo estudio. Es un juego gratuito, con un modelo de negocio basado en microtransacciones y publicidad integrada en el videojuego. Tiene una posición muy elevada en el ránking de descargas, en tanto que, en junio de 2014, sumaban más de un billón de descargas entre los dos (Webster, 2014).

Ilustración 7 Logo Temple Run 2, fuente: Imangi Studios

Ilustración 6 Captura de Temple Run 2, fuente: Imangi Studios

El juego, tiene una temática parecida a las del famoso arqueólogo que tantas películas ha protagonizado. Se avanza a través de un desplazamiento vertical, por el que el protagonista ha de sortear minas, giros cerrados, rocas y una serie de distintos obstáculos. Además, hoy en día usa la oscilación del propio dispositivo móvil, para ir alterando la posición del personaje por el camino o carril por el cual va avanzando.

Sumado a lo anterior, posee una serie de potenciadores para mejorar la experiencia de juego, que consisten en misiones diarias con el fin de hacerlo más ameno y desafiante. Sin olvidar, que en todo momento te perseguirán unos monos bastante inquietantes, que en caso de que falles varios estorbos, no dudarán es darte caza.

- **T-REX RUNNER**

Es tal la inmersión instantánea en este tipo de juegos que, Google, en septiembre de 2014, decidió lanzar el llamado T-Rex Runner. Este juego se mostraba y se sigue mostrando cuando el usuario tiene un problema en la navegación debido a la conexión a internet. Aunque también se puede usar ***chrome://dino,*** que es la dirección para acceder a él a través de Google Chrome (The new York Times, 2017).

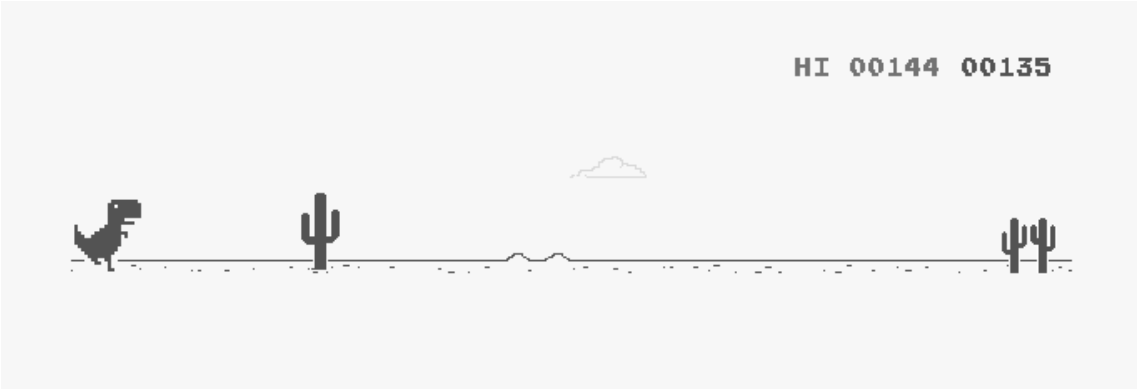

Ilustración 8 T-Rex Runner, fuente: Google

Consiste en un dinosaurio con un movimiento prácticamente rígido, el cual va saltando cactus y esquivando dinosaurios voladores. La escena se desplaza meramente en X, lo cual irá generando puntos a medida que avancemos. A su vez, la velocidad de desplazamiento irá aumentando progresivamente conforme vayamos avanzando en la partida.

- **PAC-MAN 256**

Desarrollado por Hipster Whale y 3 Sprockets y publicado por Bandai Namco Entertainment. Se lanzó como un título gratuito el 20 de agosto de 2015 para dispositivos móviles, aunque posteriormente se hizo una adaptación para Xbox, Ps4 y PC. El juego está inspirado en el fallo del nivel 256 del juego original (Matulef, 2016).

Ilustración 10 Logo Pac-Man 256, fuente: Bandai Namco Entertainment

Ilustración 9 Captura de juego de Pac-Man 256, fuente: Bandai Namco Entertainment

Respecto a la jugabilidad, no hay grandes aspectos que comentar, puesto que salvando algunas diferencias, consiste en lo que era el original PAC-MAN. Un comecocos que va huyendo/comiendo fantasmitas y recogiendo puntos a su paso. Eso sí, esta vez el mundo es de generación infinita y el escenario se va desplazando en diagonal con tendencia ascendente.

- **SPIDER-MAN UNLIMITED**

Desarrollado por Marvel, y lanzado al mercado en 2014 para plataformas Android, iPad y iPhone. Fue el primer *endless runner* del trepamuros y casi como todos los vistos en este apartado, era gratuito. Actualmente se encuentra descatalogado de todas las tiendas, de hecho, es así desde 2019 (3Djuegos, 2021).

La jugabilidad que se podía encontrar en este juego era muy similar a la que tiene el Temple Run 1 y 2.

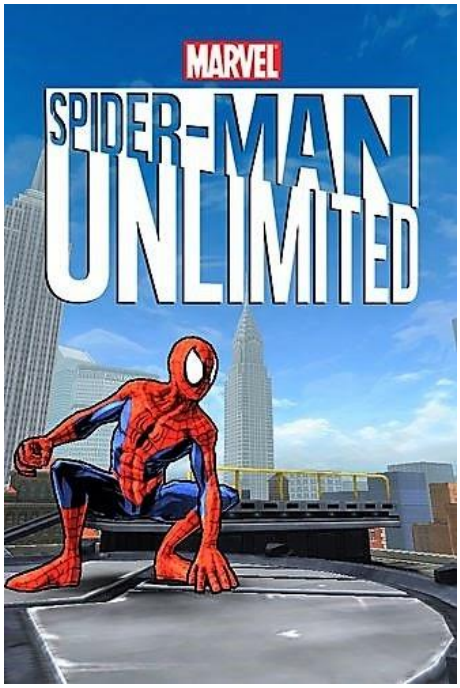

Ilustración 11 Portada Spider-Man Unlimited, fuente: Marvel Studios

Ilustración 12 Captura de juego de Spider-Man Unlimited, fuente: Marvel Studios

- **JETPACK JOYRIDE**

Desarrollado y distribuido por Halfbrick Studios, fue lanzado en la App Store el 1 de septiembre de 2011, aunque luego vería otros mercados, entre los que incluso están las videoconsolas de mesa. No fue el primer juego de éxito del desarrollador principal, ya que previamente había participado en Fruit Ninja, otro juego para móviles con mucho éxito.

El estilo de jugabilidad es muy parecido al del Flappy Bird o al del juego de Google Chrome. El personaje avanza en el eje X e intenta esquivar a su paso una serie de obstáculos que se le van apareciendo. Aunque similar a los otros dos, este no es tan sencillo y presenta multitud de fondos, enemigos, armas e incluso mejoras para el personaje durante la partida.

Ilustración 13 Portada de Jetpack Joyride, fuente: Halfbrick Studios

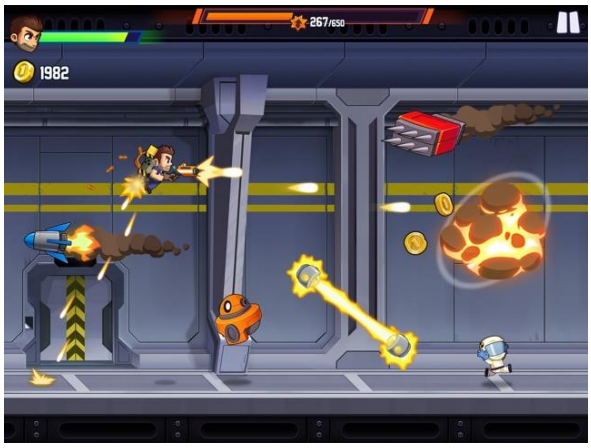

Ilustración 14 Captura de juego de Jetpack Joyride, fuente: Halfbrick Studios

3 Objetivos

El objetivo principal es el desarrollo de un videojuego 2D de tipo Runner en el motor Unity con C# con un modelo de negocio integrado y su publicación en la plataforma Play Store.
Y de aquí, podemos dividirlo en varios subobjetivos, como podría ser el poner en práctica las metodologías ágiles aprendidas a lo largo de la carrera, o el lanzarse directamente a un entorno completamente laboral, al publicar el juego directamente en un mercado abierto para todo el mundo.

El objetivo principal se puede dividir en los siguientes objetivos más concretos:

- Aprendizaje del motor Unity, y por tanto del lenguaje de programación C#. Todo el proyecto estará basado en el motor 2D de Unity.
- Utilización de las metodologías ágiles vistas a lo largo del grado académico. Todas estas herramientas serán de gran ayuda para la gestión y planificación de todo el proyecto.
- Definición e integración del modelo de negocio. Este modelo estará basado principalmente en publicidad, que se mostrará de manera estática en algunos apartados del juego, y también en forma de vídeos en otros apartados. Por ejemplo, en el caso de perder la partida, tendrás la oportunidad de seguir si visualizas el vídeo de un anuncio.
- Diseño e implementación de todos los apartados que englobe el videojuego. Desde un mundo de generación infinita, al diseño de este, pasando por el sonido, el personaje, el fondo visual e incluso por los distintos bloques que se acaban convirtiendo en un escenario que se genera infinitamente.
- Generación del fichero en formato .APK necesario para su publicación en la Play Store de Google e integración en esta misma tienda. No solo bastará con generar la .APK a través de Unity, el juego como paso necesario para su publicación y poder dar por terminado el proyecto, deberá pasar la validación de la Play Store.

4 Metodología

Se ha empleado una combinación entre una metodología ágil y un modelo de desarrollo por prototipos. Para la parte de control y gestión del proyecto se ha usado una metodología ágil, en concreto la metodología Scrum. Por otro lado, para el desarrollo software del proyecto, se ha empleado una metodología basada en modelo por prototipos. Además, para el control de versiones, se ha utilizado GitHub.

4.1 Metodología de desarrollo de software

La metodología utilizada para el desarrollo de software ha sido el **modelo por prototipos**, el cual es un modelo de desarrollo evolutivo. Consiste en construcciones rápidas y sin muchos recursos, del programa a desarrollar (Grimm, 1998).

Tiene diversas etapas, entre ellas destacamos la de diseño rápido, construcción del prototipo, desarrollo y retroalimentación y, por último, la entrega final. El desarrollo software basado en estas etapas provee varias ventajas, entre las que destacamos la celeridad a la hora de avanzar en el proyecto y, la tremenda posibilidad de reutilización de código entre los distintos prototipos generados.

Después cada uno de estos prototipos, debe de ser evaluado ya sea por el cliente o por el desarrollador, para una correcta retroalimentación. Gracias a todo esto, se refinan los requisitos de software del próximo prototipo. En caso de ser un proyecto orientado a un cliente, permite que este tenga la posibilidad de ver resultados a corto plazo.

Focalizando el modelo de prototipos hacia este proyecto, hemos de destacar que se ha utilizado para maximizar la calidad del producto final, teniendo en cuenta la limitación de tiempo que había para su desarrollo.

Se han ido generando varios prototipos, empezando por uno muy básico para entender rápidamente el funcionamiento de Unity, y acabando por otro que era un producto casi final, y el cual sirvió para entender que mecánicas eran necesarias pulir, para poder lanzar el juego a mercado.

4.2 Metodología de control y gestión del proyecto

La metodología utilizada para ejecutar el control y la gestión del proyecto se basa en una metodología ágil como Scrum.

Este tipo de método se suele aplicar a proyectos ejecutados en grupo, por lo tanto, las tareas se pueden separar bien para que todos los miembros puedan tener un trabajo independiente, y de este modo, obtengan resultados lo antes posible. Aunque este proyecto ha sido

desarrollado por una sola persona, se ha decidido incorporar este método de trabajo, debido a que ayudaría a realizar la gestión de una forma correcta y rápida.

Se ha aplicado un desarrollo iterativo, por el que se planificaron unas iteraciones temporales, teniendo en cuenta el tiempo de duración de estas, y asignándoles una carga de trabajo correspondiente al tiempo disponible. A principio de cada iteración, se hace una planificación de que tareas se van a realizar. Y al final de cada una de ellas, se hace una valoración con retrospectiva del trabajo realizado.

4.2.1 HERRAMIENTAS UTILIZADAS

Como complemento para un correcto uso de esta metodología, se han usado distintas herramientas que se van a explicar.

4.2.1.1 TRELLO

Es un software de administración de proyectos con interfaz web. La primera versión salió en 2011, y actualmente cuenta con más de 35 millones de usuarios activos.

Trello, es un tablón virtual que emplea el sistema Kanban, para el registro y la organización de tareas. Provee de un montón de funcionalidades distintas, ya sea agregar listas, comentarios, adjuntar etiquetas a las tareas, archivarlas, la posibilidad de hacer tableros compartidos, introducir imágenes, etc (Trello, 2021).

En este proyecto en concreto, se ha utilizado para planificar cada una de las iteraciones previamente comentadas en el desarrollo iterativo de la metodología ágil Scrum. Así al principio de cada iteración, se hacía una tarjeta de cada tarea, con su correspondiente comentario y su etiqueta descriptiva.

Una vez acabada cada tarea, se mueve la tarjeta al apartado de tarjetas hechas a lo largo de esa iteración. Además, el tablero posee un apartado para saber que tarjetas se están realizando en este momento. Otro, para saber posibles ideas para el futuro, y el último para saber que tareas faltan por desarrollar en la iteración actual.

Cuando finaliza la iteración, se revisa este tablero para poder sacar conclusiones de cómo ha sido el desarrollo de problemas en ese periodo de tiempo.

4.2.1.2 TOGGL TRACK

Anteriormente llamada Toggl, es una aplicación de seguimiento y control del tiempo, desarrollada por Toggl OÜ. Toggl Track, ofrece servicios de informes y seguimiento del tiempo por medio de su interfaz web (Tambur, 2013).

Tiene la capacidad de generar y gestionar equipos de trabajo, de crear un sinfín de etiquetas personalizadas para cada tarea y también ofrece la posibilidad de tener varios espacios de trabajo en una misma cuenta. Asimismo, posee integración directa con Trello a través de un plugin, facilitando de este modo su uso.

Su método de uso ha sido y es muy simple, una vez empiezas a trabajar en una tarea, inicias el contador desde el plugin que tiene Trello integrado, introduciendo las etiquetas correspondientes. Cuando termines de trabajar o termines con la tarea, pausas el contador e inicias otra, si vas a trabajar en otra distinta.

4.3 Control de versiones y repositorio

Para todo el control de versiones y como repositorio se ha utilizado GitHub. Esta plataforma, se utiliza tanto para alojar como para gestionar proyectos y sus respectivas versiones. Fue fundada hace más de 12 años, y actualmente es propiedad de Microsoft, la cual compró íntegramente GitHub por la cantidad de 7500 millones de dólares (Microsoft News Center, 2018).

Todo el control de versiones de GitHub corre a través de Git. Este es un software diseñado por Linus Torvalds y lanzado el 7 de abril de 2005. Su función es rastrear los cambios en los archivos de la computadora en la que corre, y coordinar el trabajo realizado por varias personas, en caso de que los archivos sean compartidos (Torvalds).

En definitiva, nos facilitara muchísimo el almacenamiento de todo nuestro proyecto, y también a una correcta organización y control del mismo. En parte, gracias a las facilidades que da para distinguir cada una de las versiones, entre las que podemos distinguir la posibilidad de añadir un título e incluso comentarios a cada una de esas versiones.

Por último, hay que destacar que se ha utilizado tanto por comandos a través de una máquina, como por medio de Visual Studio, el cual provee una integración directa con los programas comentados previamente, lo cual facilita su uso sobre todo para usuarios inexpertos.

5 Análisis

En este apartado, se hará un amplio recorrido, inspeccionando cada una de las facetas que hace posible el desarrollo de este proyecto. Un buen análisis puede hacer que el valor del proyecto se vea gratamente aumentado. Entre los apartados importantes se pueden destacar la especificación de requisitos, la estimación de costes, el plan de riesgos y la planificación temporal.
Además del valor añadido que le proporciona, también añade un factor de viabilidad muy importante. Toda la información obtenida de las estimaciones, son un muy buen punto de partida para saber si el proyecto puede ser desarrollado o no.
Por último, es necesario destacar que todos estos apartados se han adaptado a las necesidades y requerimientos del proyecto desarrollado, además de que existe la posibilidad de que algunos de ellos hayan sufrido pequeñas variaciones durante el proceso de desarrollo.

A continuación, se comentarán en detalle cada uno de los apartados generales desarrollado en el apartado de análisis:

En el apartado 5.1 Identificación de los requisitos se hace un paso exhaustivo por cada uno de los requisitos y funcionalidades que el videojuego debe cumplir, incluyendo tanto requisitos funcionales, que son las funcionalidades que el sistema debe cumplir para que un usuario pueda jugar correctamente, como requisitos no funcionales, que no son requisitos con los que el usuario va a interactuar directamente, pero a su vez siguen siendo propiedades necesarias para el proyecto.

A lo largo del apartado 5.2 Estudio de la viabilidad se incluyen los apartados necesarios para justificar la viabilidad del proyecto. Las secciones incluidas son: gestión de riesgos, estimación de costes (gran abanico de subapartados), nicho de mercado (público objetivo) y modelo de negocio.

Por último, en el apartado 5.3 Herramientas Software utilizadas se repasan aquellas herramientas utilizadas para el desarrollo de este videojuego, y en qué aspecto ha intervenido cada una de ellas.

5.1 Identificación de los requisitos

En esta sección se detallarán todos los requisitos de la aplicación propuesta. Se distinguen tres tipos de requisitos: obligatorios, básicos y avanzados.

Los requisitos serán identificados mediante un código alfanumérico que tendrá la siguiente estructura TIPO - NÚMERO, donde cada elemento tiene la siguiente descripción:

Nombre	Valores posibles	Descripción del elemento
TIPO	RF, RNF	RF – requisito funcional, definen las funciones de un sistema de software o sus componentes. Una función se describe como un conjunto de entradas, comportamientos y salidas. Los requisitos funcionales pueden ser: cálculos, detalles técnicos, operaciones de datos y otras funciones específicas que el sistema debería implementar. (Wiegers, 2003). RNF – requisito no funcional, describen funcionalidades que no interfieren de forma directa con el sistema. Por tanto, se refieren a todos los requisitos que no describen información a guardar, ni funciones a ejecutar, sino a características de su funcionamiento (Stellman & Greene, 2005).
NÚMERO	Número entero	Un número identificativo del requerimiento

Tabla 1 Identificación de los requisitos - Fuente: elaboración propia

Los atributos elegidos para mostrar la información de cada uno de estos requisitos serán los siguientes:

Identificador	Identificador único descrito anteriormente
Título	Una pequeña frase que encapsule el requisito
Descripción	Significado al completo de cada requisito

Tabla 2 Descripción atributos - Fuente: elaboración propia

Una vez descrita la lógica esperada, describiremos los requisitos extraídos del proyecto a desarrollar:

Identificador	RF-1
Título	Iniciar el juego
Descripción	El dispositivo debe poder cargar la escena del menú principal

Tabla 3 Requisito Funcional 1 - Fuente: elaboración propia

Identificador	RF-2
Título	Cargar el menú principal
Descripción	El menú debe poder cargarse correctamente y tener preparadas las opciones disponibles. Empezar partida, controles y salir del juego.

Tabla 4 Requisito Funcional 2 - Fuente: elaboración propia

Identificador	RF-3
Título	Mostrar controles
Descripción	El dispositivo debe poder cargar los controles y mostrar la información de estos.

Tabla 5 Requisito Funcional 3 - Fuente: elaboración propia

Identificador	RF-4
Título	Iniciar una partida desde el menú principal
Descripción	El dispositivo debe poder cargar la escena juego desde el menú principal.

Tabla 6 Requisito Funcional 4 - Fuente: elaboración propia

Identificador	RF-5
Título	Pausar la partida
Descripción	Mientras esté cargada la escena de juego, se debe proporcionar la posibilidad de pausar la partida.

Tabla 7 Requisito Funcional 5 - Fuente: elaboración propia

Identificador	RF-6
Título	Mostrar el menú de pausa
Descripción	Una vez el juego se pause, deberá aparecer el menú de pausa y mostrar las opciones disponibles: reanudar partida, mostrar controles y volver al menú principal.

Tabla 8 Requisito Funcional 6 - Fuente: elaboración propia

Identificador	RF-7
Título	Reanudar la partida
Descripción	El dispositivo debe poder reanudar la partida desde el menú de pausa.

Tabla 9 Requisito Funcional 7 - Fuente: elaboración propia

Identificador	RF-8
Título	Sistema de puntuación
Descripción	Sistema para poder mostrar el progreso en la partida del jugador. Además, será necesario para su posterior subida a la tabla de resultados de Google Play.

Tabla 10 Requisito Funcional 8 - Fuente: elaboración propia

Identificador	RF-9
Título	Mostrar el HUD
Descripción	Es necesario que cuando haya una partida en marcha, se muestre un HUD con la información de la puntuación y el botón de pausa.

Tabla 11 Requisito Funcional 9 - Fuente: elaboración propia

Identificador	RF-10
Título	Interactuar con el dispositivo en partida
Descripción	El usuario debe poder interactuar con el dispositivo para poder ejecutar las acciones necesarias.

Tabla 12 Requisito Funcional 10 - Fuente: elaboración propia

Identificador	RF-11
Título	Efecto *parallax* en el menú principal y en el juego
Descripción	El software contará con un script encargado de manejar capas infinitas y moverlas cada una a la velocidad seleccionada. Esto ayudará a mejorar el apartado visual.

Tabla 13 Requisito Funcional 11 - Fuente: elaboración propia

Identificador	RF-12
Título	Mover el personaje
Descripción	El personaje nada más iniciar una partida se moverá automáticamente hacia la derecha de forma horizontal. La cámara y la generación de mundo le seguirá.

Tabla 14 Requisito Funcional 12 - Fuente: elaboración propia

Identificador	RF-13
Título	Controlador de escena
Descripción	El juego contará con un script encargado de generar y destruir el mundo a la velocidad que el personaje avanza.

Tabla 15 Requisito Funcional 13 - Fuente: elaboración propia

Identificador	RF-14
Título	Mecánica de salto
Descripción	El usuario que esté jugando deberá tener la posibilidad de hacer saltar al personaje principal.

Tabla 16 Requisito Funcional 14 - Fuente: elaboración propia

Identificador	RF-15
Título	Mecánica de deslice
Descripción	El usuario que esté jugando deberá tener la posibilidad de hacer deslizar al personaje para evitar obstáculos.

Tabla 17 Requisito Funcional 15 - Fuente: elaboración propia

Identificador	RF-16
Título	Movimiento de enemigos
Descripción	El software tendrá un script encargado de asignar una velocidad de movimiento y un desplazamiento a cualquiera de los enemigos que vayan apareciendo.

Tabla 18 Requisito Funcional 16 - Fuente: elaboración propia

Identificador	RF-17
Título	Controlador de animaciones
Descripción	El software poseerá un script con la capacidad de ir fluyendo entre los distintos estados por los que el personaje principal vaya pasando.

Tabla 19 Requisito Funcional 17 - Fuente: elaboración propia

Identificador	RF-18
Título	Mostrar anuncios
Descripción	Se debe mostrar anuncios en determinados momentos de la ejecución del juego.

Tabla 20 Requisito Funcional 18 - Fuente: elaboración propia

Identificador	RF-19
Título	Salir del juego
Descripción	El usuario debe de tener la posibilidad de cerrar el juego sin problema.

Tabla 21 Requisito Funcional 19 - Fuente: elaboración propia

Identificador	RNF-1
Título	El juego debe estar subido a la Play Store
Descripción	La aplicación debe pasar los filtros de validación de Google, para que se pueda descargar desde la tienda de aplicaciones.

Tabla 22 Requisito No Funcional 1 - Fuente: elaboración propia

Identificador	RNF-2
Título	Proporcionar logros al usuario
Descripción	El juego debe de ser capaz de generar logros que podrá ver a través de Google Play.

Tabla 23 Requisito No Funcional 2 - Fuente: elaboración propia

Identificador	RNF-3
Título	Usabilidad
Descripción	El juego ha de ser usable sea cual sea el usuario. Debe tener una curva de aprendizaje correcta.

Tabla 24 Requisito No Funcional 3 - Fuente: elaboración propia

Identificador	RNF-4
Título	Rendimiento
Descripción	El juego debe funcionar con un rendimiento adecuado independientemente del dispositivo donde se juegue.

Tabla 25 Requisito No Funcional 4 - Fuente: elaboración propia

Identificador	RNF-5
Título	Estabilidad
Descripción	La aplicación tiene que mantener un mismo funcionamiento sea cual sea el sistema Android que la corra. No se pueden generar cambios en el desarrollo del juego dependiendo de las características del sistema.

Tabla 26 Requisito No Funcional 5 - Fuente: elaboración propia

Identificador	RNF-6
Título	Escalabilidad
Descripción	El juego debe de ser pensado y programado, de forma que sea extensible a lo largo del tiempo, sin perder sistemas de puntuación y sin modificar por completo la experiencia de usuario.

Tabla 27 Requisito No Funcional 6 - Fuente: elaboración propia

5.2 Estudio de la viabilidad

A lo largo de este punto, se hará un exhaustivo análisis a todos los apartados necesarios para poder estudiar la viabilidad del proyecto. Estos apartados se podrían dividir en dos grandes bloques, uno asociado a los problemas o riesgos que podrían aparecer durante el desarrollo del proyecto, y otro, encargado de calcular el coste de este e intentar rentabilizarlo.

Toda esta información, aparte de detallar si es factible o no, nos dará un activo de gran valor a la hora de intentar vender o distribuir el videojuego.

Este estudio se divide en cinco partes:

- En primer lugar, se desarrollará el plan de gestión de riesgos inherentes al proyecto y, por ende, a su desarrollo.
- A continuación, se hará una estimación de todos los tipos de costes involucrados directa o indirectamente en el proyecto.
- Seguidamente, se investigarán todos los datos necesarios y se explicará cuál es el nicho de mercado donde este videojuego espera tener su espacio.
- Para finalizar y utilizando todos los datos anteriores, se relatará cuál es el modelo de negocio adecuado a seguir, además de cómo se llevará a cabo.

5.2.1 GESTIÓN DE RIESGOS

Para la gestión de riesgos, se han identificado todos los posibles riesgos que pueden surgir en el transcurso y desarrollo del proyecto. Posteriormente, y una vez todos los riesgos queden acotados, han sido analizados y catalogados por probabilidades y consecuencias; después, se han buscado posibles métodos de prevención, de minimización y de contingencia para cada uno de estos posibles riesgos; y por último, se ha intentado encontrar distintos signos para monitorizar e intentar evitar que estos riesgos lleguen a causar estragos (Sommerville R. , 1998).

5.2.1.1 Identificación

En este subapartado se van a exponer todos los posibles riesgos que podrían afectar al proyecto, asociados a una serie de categorías que agrupan cada uno de los riesgos en un mismo entorno. Así que, para empezar, vamos a hacer una pequeña introducción de los tipos de riesgo que se podrían representar con un ejemplo lo más claro posible:

TIPO DE RIESGO	POSIBLE RIESGO
Tecnología	La Base de Datos utilizada no puede procesar muchas transacciones por segundo como se espera.
Personas	Es imposible seleccionar personal con las habilidades requeridas para el proyecto.
Organizacional	Los problemas financieros en la organización causan reducciones en el presupuesto del proyecto.
Herramientas	Las herramientas CASE no se pueden integrar.
Requerimientos	Se proponen cambios en los requerimientos que suponen rehacer el diseño.
Estimación	El tiempo requerido para desarrollar el software esta infraestimado.

Tabla 28 Definición categorías elegidas. Fuente: (Sommerville I. , 2005)

Una vez explicados los distintos tipos de riesgos que podremos encontrar, vamos a hacer una búsqueda de los posibles riesgos del proyecto:

TIPO DE RIESGO	POSIBLE RIESGO
Tecnología	Fallo de hardware.
	Componentes técnicos con vulnerabilidades en seguridad.
	Pérdida de datos.
	Ataque a los servidores externos (detallar más).

Tabla 29 Riesgos tecnológicos identificados - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO
Personas	Fallecimiento del desarrollador.
	Fallecimiento de un familiar.
	Enfermedad del desarrollador.
	Enfermedad de un familiar.
	La falta de conocimiento en algún área del proyecto retrasa este mismo.
	Pérdida de tiempo por necesidad de buscar otro trabajo remunerado.
	Falta de motivación del desarrollador.
	Abandono del proyecto.
	Falta de ideas durante el proceso de diseño.

Tabla 30 Riesgos relacionados con personas identificados - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO
Organización	No disposición de fondos suficientes para realizar correctamente el proyecto.
	El mercado entra en crisis, y por tanto la demanda baja cuantiosamente.
	El sector informático se desploma y baja la cotización del proyecto. Por lo que el valor de mercado desciende, pero los costes se mantienen.

Tabla 31 Riesgos de organización identificados - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO
Herramientas	Encarecimiento de la utilización de Unity para proyectos como el desarrollado.
	Encarecimiento de las herramientas que provee la Play Store.
	El gobierno prohíbe internet, o en su defecto las aplicaciones basadas en microtransacciones.
	Fallos en algunas de las herramientas que posee Unity.
	Pérdida de información por parte del repositorio GitHub.

	Fallos o pérdida de datos en la Play Store.

Tabla 32 Riesgos asociados a las herramientas identificados - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO
Requerimientos	Se proponen cambios en los requerimientos que suponen rehacer el diseño.
	Se proponen cambios en los requerimientos que suponen rehacer funcionalidades.
	Necesidad de cambiar requisitos no funcionales.
	Rediseño de la base de datos y sus requerimientos.

Tabla 33 Riesgos relacionados con los requerimientos identificados- Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO
Estimación	El tiempo requerido para desarrollar el software está infraestimado.
	Fallo en la estimación de horas del diseño visual del proyecto.
	El coste del proyecto es mayor del previsto.
	Coste humanitario superior del previsto.
	Coste de software y hardware superior del previsto.

Tabla 34 Riesgos de estimación identificados - Fuente: elaboración propia

5.2.1.2 Análisis

En el siguiente subapartado se asociará una probabilidad y los efectos de cada uno de los posibles riesgos.

Esta labor no es fácil y depende mucho de la experiencia y el juicio del estimador. Por ello, para facilitar la tarea, se divide la probabilidad en intervalos y los efectos en categorías, de la siguiente forma:

- Probabilidad: Muy Baja (<10%), Baja (10-25%), Moderada (25-50%), Alta (50-75%) y Muy Alta (>75%).
- Efectos: Catastrófico, Serio, Tolerable o Insignificante.

Además, es importante destacar que están ordenados por las categorías que hemos visto previamente. Asimismo, cada categoría está organizada por efectos, de más a menos graves.

En caso de que haya dos riesgos en la misma categoría, que supongan los mismos efectos, aparecerá antes en la tabla la que mayor probabilidad tenga.

Una vez detallada la información a visualizar, vamos a proceder al análisis:

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Tecnología	Pérdida de datos.	Baja	Serio
	Componentes técnicos con vulnerabilidades en seguridad.	Alta	Tolerable
	Fallo de hardware.	Baja	Tolerable
	Ataque a los servidores externos.	Baja	Tolerable

Tabla 35 Análisis riesgos tecnológicos - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Personas	Abandono del proyecto.	Baja	Catastrófico
	Fallecimiento del desarrollador.	Muy Baja	Catastrófico
	La falta de conocimiento en algún área del proyecto retrasa este mismo.	Moderada	Serio
	Falta de ideas durante el proceso de diseño.	Moderada	Serio
	Enfermedad del desarrollador.	Baja	Serio
	Enfermedad de un familiar.	Baja	Serio
	Falta de motivación del desarrollador.	Baja	Serio
	Fallecimiento de un familiar.	Muy Baja	Serio
	Pérdida de tiempo por necesidad de buscar otro trabajo remunerado.	Moderada	Tolerable

Tabla 36 Análisis riesgos relacionados con las personas - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Organización	El mercado entra en crisis, y por tanto la demanda baja cuantiosamente.	Baja	Serio
	El sector informático se desploma y baja la cotización del proyecto. Por lo que el valor de mercado desciende, pero los costes se mantienen.	Muy baja	Serio
	No disposición de fondos suficientes para realizar correctamente el proyecto.	Muy Baja	Tolerable

Tabla 37 Análisis riesgos asociados a la organización - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Herramientas	El gobierno prohíbe internet, o en su defecto las aplicaciones basadas en microtransacciones.	Muy Baja	Catastrófico
	Pérdida de información por parte del repositorio GitHub.	Muy Baja	Catastrófico
	Encarecimiento de la utilización de Unity para proyectos como el desarrollado.	Baja	Serio
	Encarecimiento de las herramientas que provee la Play Store.	Baja	Serio
	Fallos en algunas de las herramientas que posee Unity.	Muy Baja	Serio
	Fallos o pérdida de datos en la Play Store.	Muy Baja	Serio

Tabla 38 Análisis riesgos relacionados con las herramientas - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Requerimientos	Se proponen cambios en los requerimientos que suponen rehacer el diseño.	Alta	Tolerable
	Se proponen cambios en los requerimientos que suponen rehacer funcionalidades.	Alta	Tolerable
	Necesidad de cambiar requisitos no funcionales.	Alta	Tolerable
	Rediseño de la base de datos y sus requerimientos.	Moderada	Tolerable

Tabla 39 Análisis riesgos de requerimientos - Fuente: elaboración propia

TIPO DE RIESGO	POSIBLE RIESGO	PROBABILIDAD	EFECTOS
Estimación	El tiempo requerido para desarrollar el software está infraestimado.	Alta	Serio
	Fallo en la estimación de horas del diseño visual del proyecto.	Alta	Serio
	El coste del proyecto es mayor del previsto.	Moderada	Serio
	Coste humanitario superior del previsto.	Baja	Serio
	Coste de software y hardware superior del previsto.	Baja	Serio

Tabla 40 Análisis riesgos de estimación - Fuente: elaboración propia

5.2.1.3 Planificación

A continuación, vamos a ver el apartado más importante, consiste en establecer estrategias para abordar los riesgos identificados y analizados previamente. Por lo tanto, vamos a añadir siempre que sea posible, una estrategia de prevención, otra de minimización y un plan de contingencia para cada uno de los posibles riesgos. Vamos a desglosar cada una de ellas:

- **Prevención**: ideas o estrategias para intentar evitar la llegada del posible riesgo.
- **Minimización**: en caso de que el riesgo ocurra, estrategia para procurar disminuir las consecuencias que este pueda ocasionar.
- **Plan de contingencia**: si el riesgo ocurre y sin posibilidad de minimizar daños, estrategia para cortar los estragos con celeridad.

Así quedaría cada riesgo con sus estrategias:

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Tecnología	Fallo de hardware.	**Prevención:** tener más cuidado y aplicar las mejores medidas de protección posibles. **Minimización:** reparar el ordenador lo antes posible o buscar otro método con el que poder trabajar sin problemas. **Plan de Contingencia:** tener material de repuesto.
	Componentes técnicos con vulnerabilidades en seguridad.	**Prevención:** tener cuidado sobre dónde y cómo se desarrolla el proyecto. **Minimización:** contener la sobreexposición. **Plan de Contingencia:** no hay
	Pérdida de datos.	**Prevención:** monitorizar los lugares donde almacenamos información. **Minimización:** limitar el acceso. **Plan de Contingencia:** tener copias de seguridad activas de todo.
	Ataque a los servidores externos.	**Prevención:** no hay. **Minimización:** no hay. **Plan de Contingencia:** tener copias de seguridad de la información volcada a servidores externos al proyecto.

Tabla 41 Planificación riesgos tecnológicos - Fuente: elaboración propia

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Personas	Fallecimiento del desarrollador.	**Prevención:** ser más precavido con el cuidado personal. **Minimización:** consumo de farmacéuticos o consultar el problema con un profesional. **Plan de Contingencia:** no hay.
	Fallecimiento de un familiar.	**Prevención:** intentar concienciar y cuidar de los familiares cercanos **Minimización:** consumo de farmacéuticos o consultar el problema con un profesional. **Plan de Contingencia:** no hay.
	Enfermedad del desarrollador.	**Prevención:** ser más precavido con el cuidado personal. **Minimización:** consumo de farmacéuticos o consultar el problema con un profesional. **Plan de Contingencia:** estar equipado con una serie de fármacos básicos de emergencia.
	Enfermedad de un familiar.	**Prevención:** intentar concienciar y cuidar de los familiares cercanos **Minimización:** consumo de farmacéuticos o consultar el problema con un profesional. **Plan de Contingencia:** estar equipado con una serie de fármacos básicos de emergencia.
	Falta de ideas durante el proceso de diseño.	**Prevención:** evitar saturarse durante el proceso.

		Minimización: buscar ayuda o inspiración en agentes externos. **Plan de Contingencia:** aprovechar momentos de inspiración para crear distintos diseños para el futuro.
	La falta de conocimiento en algún área del proyecto retrasa este mismo.	**Prevención:** dedicar más horas al proyecto. Evitar distracciones a la hora de trabajar. **Minimización:** reorganizar al horario buscando sacar más provecho a las horas. **Plan de Contingencia:** recurrir a aliados para pedir ayuda, intentar sacar más horas de estudia.
	Pérdida de tiempo por necesidad de buscar otro trabajo remunerado.	**Prevención:** preservar el trabajo actual en medida de lo posible. **Minimización:** ahorro de dinero para intentar subsistir. **Plan de Contingencia:** dejar de lado el desarrollo hasta poder conseguir un trabajo remunerado.
	Falta de motivación del desarrollador.	**Prevención:** llevar buenos hábitos mentales. **Minimización:** intentar animarse en media de lo posible. **Plan de Contingencia:** tomar un pequeño descanso en medida de lo posible hasta que la motivación vuelva.

	Abandono del proyecto.	**Prevención:** mantener la motivación lo más alta posible. **Minimización:** reorganizar el trabajo intentando no saturar. **Plan de Contingencia:** no hay

Tabla 42 Planificación riesgos relacionados con personas - Fuente: elaboración propia

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Organización	No disposición de fondos suficientes para realizar correctamente el proyecto.	**Prevención:** hacer una buena estimación de costes. **Minimización:** intentar reducir los costes de algunas cosas prescindibles. **Plan de Contingencia:** no hacer gastos innecesarios ni sobrepasar los presupuestos de no ser estrictamente necesario.
	El mercado entra en crisis, y por tanto la demanda baja cuantiosamente.	**Prevención:** realizar un buen estudio de mercado antes de empezar el proyecto. **Minimización:** no hay. **Plan de Contingencia:** estudiar debidamente el mercado y las posibles subidas o bajadas de demanda, estando preparados para cualquiera de las dos posibilidades.
	El sector informático se desploma y baja la cotización del proyecto. Por lo que el valor de mercado desciende, pero los costes se mantienen.	**Prevención:** no hay. **Minimización:** no hay. **Plan de Contingencia:** tener modelos de negocio alternativos.

Tabla 43 Planificación riesgos asociados a la organización - Fuente: elaboración propia

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Herramientas	Encarecimiento de la utilización de Unity para proyectos como el desarrollado.	**Prevención:** estudiar futuros movimientos de Unity. **Minimización:** intentar buscar algún patrocinador que financie el proyecto. **Plan de Contingencia:** tener herramientas alternativas.
	Encarecimiento de las herramientas que provee la Play Store.	**Prevención:** estudiar futuros movimientos de Google. **Minimización:** no hay. **Plan de Contingencia:** tener herramientas alternativas.
	El gobierno prohíbe internet, o en su defecto las aplicaciones basadas en microtransacciones.	**Prevención:** no hay **Minimización:** no hay **Plan de Contingencia:** buscar otro modelo de negocio.
	Fallos en algunas de las herramientas que posee Unity.	**Prevención:** no hay. **Minimización:** no hay. **Plan de Contingencia:** buscar otro software utilizable.
	Pérdida de información por parte del repositorio GitHub.	**Prevención:** copia de seguridad diaria de todas las ramas en un repositorio local, si se borran las ramas se pueden restaurar con una copia local. **Minimización:** usar un repositorio local de uno de nuestros ordenadores para restaurar los datos. **Plan de Contingencia:** restaurar una copia de seguridad lo antes posible.

	Fallos o pérdida de datos en la Play Store.	**Prevención:** copia de seguridad diaria de toda la información volcada en Google. **Minimización:** usar un repositorio local para restaurar los datos. **Plan de Contingencia:** restaurar una copia de seguridad lo antes posible.

Tabla 44 Planificación riesgos relacionados con las herramientas - Fuente: elaboración propia

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Requerimientos	Se proponen cambios en los requerimientos que suponen rehacer el diseño.	**Prevención:** intentar detallar al máximo los requerimientos. **Minimización:** no hay. **Plan de Contingencia:** estar abierto a cambios.
	Se proponen cambios en los requerimientos que suponen rehacer funcionalidades.	**Prevención:** intentar detallar al máximo los requerimientos. **Minimización:** no hay. **Plan de Contingencia:** estar abierto a cambios.
	Necesidad de cambiar requisitos no funcionales.	**Prevención:** intentar detallar al máximo los requisitos. **Minimización:** no hay. **Plan de Contingencia:** estar abierto a cambios.
	Rediseño de la base de datos y sus requerimientos.	**Prevención:** intentar detallar al máximo la base de datos de Google. **Minimización:** no hay. **Plan de Contingencia:** estar abierto a cambios.

Tabla 45 Planificación riesgos de requerimientos - Fuente: elaboración propia

TIPO DE RIESGO	RIESGO	ESTRATEGIA
Estimación	El tiempo requerido para desarrollar el software está infraestimado.	**Prevención:** estimar lo más detenidamente posible. **Minimización:** dedicar más horas al proyecto. **Plan de Contingencia:** pedir un extra de tiempo para el desarrollo.
	Coste humanitario superior del previsto.	**Prevención:** estimar lo más detenidamente posible. **Minimización:** no hay. **Plan de Contingencia:** buscar ayuda o intentar contratar a alguien.
	Fallo en la estimación de horas del diseño visual del proyecto.	**Prevención:** estimar lo más detenidamente posible. **Minimización:** dedicar más horas al proyecto. **Plan de Contingencia:** pedir un extra de tiempo para el diseño.
	El coste del proyecto es mayor del previsto.	**Prevención:** estimar lo más detenidamente posible. **Minimización:** no hay. **Plan de Contingencia:** buscar ayuda o financiación sea como sea.

	Coste de software y hardware superior del previsto.	**Prevención:** dedicar tiempo a realizar una buena estimación de costes y al estudio del estado actual de los softwares a utilizar. **Minimización:** ver la opción de migrar a otros softwares gratuitos o de menores costes. **Plan de Contingencia:** uso de freeware (softwares gratuitos), previendo que en un futuro se pueda cambiar de software, bien por costes o por funcionalidad.

Tabla 46 Planificación riesgos de estimación - Fuente: elaboración propia

5.2.1.4 Monitorización y control

Para finalizar con la gestión de riesgos, se propondrán indicadores que puedan hacer pensar que la probabilidad o los efectos de un riesgo han cambiado. Es el paso del que se parte para realizar una revisión de riesgos, chequeando estos identificadores potenciales.

Al igual que en apartados anteriores, los riesgos se han categorizado por tipo de riesgo.

Serían los siguientes:

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Tecnología	Ordenador empieza a apagarse de forma inesperada.
	Alguno de los dispositivos de hardware empieza reiniciarse cuando no tiene que hacerlo.
	Problemas al recuperar archivos de datos de alguno de los sistemas en los que se desarrolle el proyecto.

Tabla 47 Monitorización riesgos tecnológicos - Fuente: elaboración propia

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Personas	Pérdida del apetito o el sueño en el día a día del desarrollador.
	Disminución repentina del estado de salud de algún familiar.
	Exceso de trabajo externo al proyecto.
	Dificultades para desarrollar algún área.

	Cada día resulta más complicado seguir con el proyecto debido a la falta de ganas.
	Cada vez se tarda más en diseñar cada apartado del juego.

Tabla 48 Monitorización riesgos relacionados con personas- Fuente: elaboración propia

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Organización	El presupuesto inicial empieza a acabarse antes de lo previsto.
	Cada vez se pagan menos por estos proyectos, el mercado empieza a tambalearse.

Tabla 49 Monitorización riesgos asociados a la organización - Fuente: elaboración propia

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Herramientas	Unity anuncia cambios de políticas en el uso de su aplicación.
	Google anuncia cambios de políticas en el uso de su aplicación.
	Algún partido pone sobre la mesa este estilo de juegos adictivos, para discutir sobre su legalidad.
	Distintos ataques cibernéticos a plataformas similares a GitHub o la Play Store.
	Disminución progresiva de los ingresos proporcionados por Unity Ads.

Tabla 50 Monitorización riesgos relacionados con las herramientas usadas - Fuente: elaboración propia

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Requerimientos	Los usuarios se quejan de alguna de las funcionalidades desarrolladas.
	Se genera algún problema siguiendo alguno de los flujos previstos.
	Se comienza a ver funcionalidades que no tienen sentido con el diseño inicial.
	Los usuarios reportan problemas con las estadísticas de Google.

Tabla 51 Monitorización riesgos de requerimientos - Fuente: elaboración propia

TIPO DE RIESGO	IDENTIFICADORES POTENCIALES
Estimación	Se va retrasando más de lo debido cada iteración de desarrollo.
	Hay código ya desarrollado que se deshecha.
	Algunos diseños empiezan a llevarse más intentos de los debidos, incluso se vuelve atrás en alguno de ellos.
	Respecto a los costes, subida de la luz o el agua.

Tabla 52 Monitorización riesgos de estimación - Fuente: elaboración propia

5.2.2 ESTIMACIÓN DE COSTES

En este apartado, se va a detallar y enseñar todos los distintos tipos de costes que engloba el proyecto. Estos han sido elegidos en función del tipo de proyecto a desarrollar, en este caso un producto software. Cada uno de ellos es vital para que el proyecto salga adelante y un mal cálculo de alguno de ellos, supondría una reestimación del esfuerzo requerido para llevar a cabo ese componente en concreto, reestimación que tendría un impacto directo y francamente negativo sobre el futuro del proyecto.

Se empezará por los componentes de costes, donde se verán: costes de hardware y software, costes de viaje, costes de esfuerzo, otros costes y por último, el gasto total.

Una vez toda esa información esté acotada y, debido a la inexperiencia en el desarrollo y gestión de este tipo de proyectos, se pasará a utilizar dos técnicas para obtener la estimación de costes, del software en cuestión. En concreto, se verá la ley de Parkinson, con la que se conseguirá llegar a datos significativos sobre el esfuerzo necesario y, la técnica Pricing to Win, con la que se intentará sacar números reales de impresiones necesarias para conseguir beneficios.

5.2.2.1 Componentes de coste

En este apartado se van a relatar todos los componentes de coste que pueden llegar a afectar al desarrollo del proyecto, que incluyen los costes de hardware, de software, de viajes o incluso de esfuerzo o recursos de aprendizaje. Asimismo, se detallará el coste mensual total, que hay que tener en cuenta para poder hacer las estimaciones y los cálculos que vienen en los apartados a continuación de este. Por último, hay que tener en cuenta que el tiempo de desarrollo está estimado en dos meses y que, por lo tanto, todos los cálculos de coste total de cada uno de los componentes están basados en esa estimación temporal.

Vamos a empezar con los **costes de esfuerzo**, sueldos y gastos relacionados con ellos. Son los que más importancia tienen puesto que son los que mayor cantidad suponen.

Pasemos a la tabla con los datos y la información explicativa:

Costes de esfuerzo		
Descripción	**Coste por integrante mensual**	**Coste total**
Sueldo ingeniero	1451,7€	2903,4€
Gastos seguros y seguridad social	114,3€	228,6€
IRPF	234€	468€
TOTAL	1800€	3600€

Tabla 53 Costes de esfuerzo - Fuente: elaboración propia

- **Sueldo ingenieros del proyecto:** un ingeniero senior del sector puede llegar a cobrar alrededor de 7200€/mes. Dado que en este proyecto no hay ingenieros con tal experiencia, el sueldo de un ingeniero recién formado oscila sobre los 1800€ brutos. Descontando los gastos en seguridad social y IRPF, quedaría un sueldo neto de 1451,7€ (Indeed, 2021).
- **Gastos seguros y seguridad social:** se compone de tres partidas: la cotización por contingencias comunes (4,70% del sueldo bruto), la cotización por formación (0,10%) y la cotización por desempleo (1,55% en contratos indefinidos, 1,60% en contratos temporales), lo que en total supone un 6,35% del sueldo bruto. Teniendo en cuenta el sueldo mencionado anteriormente, el gasto en seguridad social de un empleado sería de 114,3€.
- **IRPF:** suele rondar el 13% para sueldos de 2000€. Por lo tanto 234€ (infoautonomos, 2019).

Por lo tanto, según los gastos asociados a la contratación de un ingeniero junior en relación con los estándares salariales en España, estaríamos hablando de unos 1800€/mes, una cantidad moderada, pero a considerar para saber cuánto puede costar recuperar la inversión.

A continuación, los **costes de hardware y software**. Estos dos son importantes debido al proyecto que se está relatando. Sin una máquina donde programar, no hay programa.

Empecemos con la tabla y con sus posteriores explicaciones:

Costes de hardware y software		
Descripción	**Coste pago único**	**Coste mensual**
Coste de licencia de software	32,19€	19,99€
Costes de hardware	1.511,78€	0€
TOTAL	1.551,76€	19,99€

Tabla 54 Costes de hardware y software - Fuente: elaboración propia

- **Coste de Licencia de software**: es necesario desglosar de dónde vienen esos 32,19 euros y los 19,99 euros mensuales, es por ello, que vamos a ver las distintas licencias de uso y el gasto que conllevan cada una de ellas.
  El coste de registrarse como desarrollador en la Play Store es de 25 dólares (21,20 euros). Una vez dado de alto como desarrollador, podrás subir todo el contenido que quieras, además de ser un pago único (Google, 2021).
  Unity no tiene coste alguno, siempre y cuando el proyecto genere menos de 100.000 dólares (85.000 euros) al año. En caso de que se supere, deberemos optar por otro tipo de licencia dentro de Unity. La licencia más cara de Unity son 200 dólares (169,44 euros) al mes (Unity, 2021).
  Sin embargo, el coste de Procreate (herramienta para el diseño que veremos posteriormente), es de 10,99 euros en un pago único (Apple, 2021). Y el coste de la licencia de Adobe Illustrator (otra herramienta de diseño que se verá posteriormente), es de 19,99 euros al mes. Puesto que han sido dos meses de desarrollo, el coste de Adobe asciende a 39,98 euros (Alice, 2020).
  El coste de todas las herramientas de gestión utilizadas es igual a 0. Trello, Toggl y GitHub, son herramientas gratuitas.

- **Coste del Hardware**: el coste total es de 1.511,78 euros, y se necesita disponer de un ordenador en el que instalar el software requerido (además de donde poder trabajar con él), un sistema de copia de seguridad para estar protegido ante los desastres, y varios dispositivos Android donde probar todas las versiones que se vayan produciendo del videojuego, tanto antes como después de su puesta en producción. De la misma manera que con las licencias de software, hay que desglosar el coste total para poder entender de dónde proviene.
  El desarrollo y las copias de seguridad podemos hacerlo en el mismo ordenador, aunque sea en distintos discos duros, por lo que el coste de un pc apto para esto tendría un valor de 982,79 euros (PcComponentes, 2021).
  Asimismo, se necesitan dos dispositivos con Android, una tableta y un teléfono móvil, ambos no tienen por qué ser dispositivos de gama alta, con ser de gama media cubrirían los requerimientos.
  Por lo tanto, un teléfono móvil de gama media lo podemos encontrar por 229 euros (Amazon, 2021), mientras que una tableta de gama media y con Android instalado, se puede obtener por 299,99 euros (Amazon, 2021).

El total de ambos pagos viene dado de la suma del coste de cada uno de los instrumentos descritos. Como gasto mensual solo habría que pagar la licencia Adobe Illustrator (19,99 x 2 meses). Por otro lado, como gasto único habría que tener en cuenta el siguiente cálculo: 982,79 (coste pc) + 229 (coste teléfono) +299,99 (coste tableta) + 32,19 (suma de Procreate más el coste para poder ser desarrollador de Android) = 1.551,76€.

Una vez especificado ambos costes, es importante destacar algo que se ha mencionado previamente. Aunque este componente de costes no es realmente elevado, es totalmente esencial para el desarrollo del software. Sin un hardware donde programar el juego y sin al menos un dispositivo Android donde probar el resultado, es imposible llevar a cabo el proyecto.

Otro de los componentes de coste, serían los **costes de viaje**.

Debido a las condiciones extraordinarias de este 2020, todo el trabajo será realizado desde casa. Con lo que el coste total en viajes o desplazamientos es igual a cero. Todo el seguimiento del proyecto se hará de forma semanal a través de Google Meet.

Soluciones especiales a problemas extraordinarios, adaptación al medio. Como el coste va a ser cero, no hay nada por aportar que sume.

En cuanto a **costes de aprendizaje**, todos los recursos utilizados son gratuitos, por lo tanto, no hay coste monetario, lo único necesario es coste temporal que se traduce en costes de esfuerzo que se pueden asumir dentro del sueldo mensual.

Por último, tenemos **otros costes** donde encapsularemos el resto de gastos que por su naturaleza, no entra en ninguno de los componentes previos. Veamos la tabla y su conclusión:

Otros costes		
Integrantes	**Descripción**	**Gasto Mensual**
Manuel Font Rodríguez	Gastos de vivienda, alquiler, luz y agua. A su vez, internet para seguir desarrollando el proyecto.	235€
TOTAL TIEMPO DESARROLLO		470€

Tabla 55 Otros costes - Fuente: elaboración propia

Al vivir en una casa compartida, los gastos son reducidos parcialmente. Del mismo modo que con el resto de los componentes, éste es importante para poder calcular el gasto total. Una vez calculado estos gastos, podremos sacar conclusiones.

Como conclusión, tenemos la suma total de todos los componentes desarrollados con anterioridad. Veamos la tabla:

Gasto total		
Integrantes	**Gasto Mensual**	**Gasto Total (2 meses + pagos únicos)**
Manuel Font Rodríguez	2054,99€	5661,74€

Tabla 56 Gasto total - Fuente: elaboración propia

Por lo tanto, el coste del proyecto se ha estimado en 5661,74 euros. Este valor sale de la suma de los gastos de las dos mensualidades, más los gastos de pago único para poder desarrollar el proyecto.

Con esta información, se puede empezar a sacar cuentas para ver si el proyecto es realmente plausible o rentable, e incluso compararlo con otras técnicas de estimación para obtener más datos sobre la viabilidad y la rentabilidad de este. Habrá que recordar estos datos para los siguientes apartados.

5.2.2.2 Ley de Parkinson

"El trabajo se expande hasta llenar el tiempo disponible para que se termine". (Parkinson, 1957)

Así es como se define la ley de Parkinson. Esta ley consiste en incrementar la productividad estableciendo límites, en nuestro caso plazos, de modo que podamos aprovechar el tiempo del que disponemos para presentar un trabajo de la mayor calidad posible.

Dado que nuestro límite es el tiempo, reducir tareas en subtareas puede ayudar aumentar la productividad. Una vez dividida la tarea, debemos priorizar las subtareas más importantes. Una forma es iniciarlas en orden de prioridad. Al seleccionar primero las cosas importantes, evitamos el riesgo de presentar algo de mala calidad.

Esta estimación consiste en calcular el esfuerzo necesario como el producto del número de recursos humanos, por el número de meses planificados de desarrollo. Si esto lo aplicáramos a este proyecto, se podría ver que tendríamos un esfuerzo de **dos personas/mes**, resultado que viene de tener un empleado durante dos meses.

Esto multiplicado por el coste de una persona al mes, nos dará el coste total del proyecto. Que según hemos calculado en el apartado anterior (**2035€/miembro**) nos da como resultado lo siguiente:

$$Coste\ total\ del\ proyecto\ =\ 2\ \frac{personas}{mes} \times\ 2035€\ \frac{persona}{mes}\ =\ \mathbf{4070}€$$

Según la ley de Parkinson, el coste de esfuerzo sería de **4070 €.**

5.2.2.3 Pricing To Win

Pricing To Win se basa en el hecho de que el coste del proyecto depende de lo que el cliente esté dispuesto a pagar. El principal beneficio es que la misma empresa promotora es la que obtiene el contrato (Amplio, 2018).

En el caso de que un cliente quisiera desarrollar un software con estas características y un presupuesto menor, podríamos recortar salarios (algo que no es rentable porque perderíamos dinero) o podríamos recortar el tiempo de desarrollo y ajustar las especificaciones para reflejar el tiempo que tenemos. Esta es la medida utilizada en Pricing to Win.

Una vez firmado el contrato, se acordará con el cliente una especificación detallada del proyecto detallando las funciones básicas que se van a entregar. También se proporciona un presupuesto para otras funciones. El cliente ya tiene un software en funcionamiento que luego no se ve aumentado por otros requisitos. Estos requisitos adicionales, a su vez, pueden conducir a un presupuesto futuro.

Si intentamos aplicar todo a esto al proyecto que se está describiendo, veremos que es complicado hacer estimaciones reales debido a las condiciones que el mercado impone.

La autofinanciación y los medios propios se mantienen como las principales herramientas para subsistir en el día a día. Ambas opciones son las más socorridas, si tenemos en cuenta que la siguiente alternativa, la liquidez ligada a un acuerdo con un *Publisher*, solo representa al 18% de las compañías (DEV - Asociación Española de Empresas Productoras, 2020).

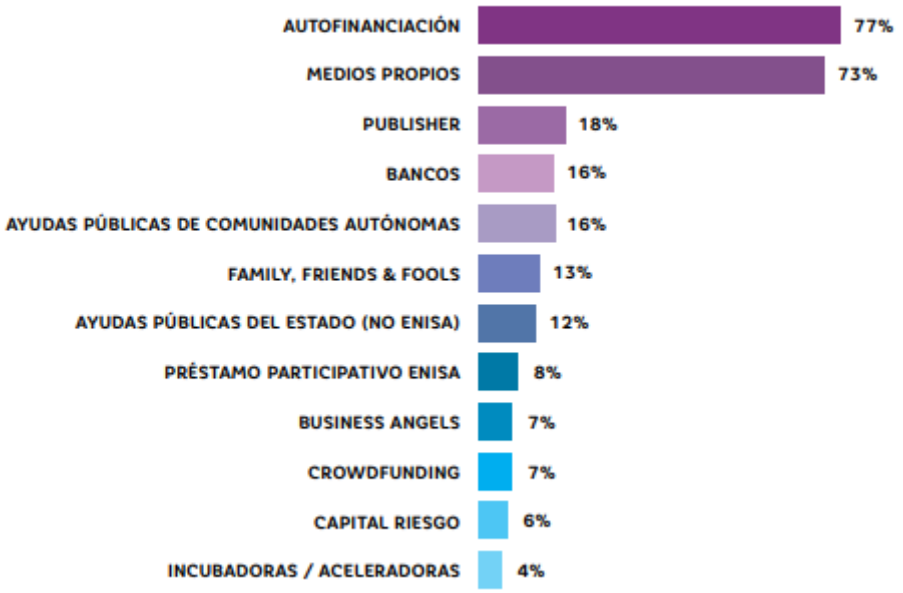

Ilustración 15 Porcentajes financiación videojuegos - Fuente: (DEV - Asociación Española de Empresas Productoras, 2020)

Con esta información citada, y observando las grandes dificultades para la financiación externa y después de revisar las cuantías de estas, se va a hacer un ejercicio donde se va a suponer que nosotros mismos somos una empresa dispuesta a desarrollar este software. Debido a que en el supuesto, seríamos una empresa medianamente novel en el sector y no sabríamos realmente el coste de desarrollar el producto (coste que se ha calculado arriba con la Ley de Parkinson), estaríamos dispuestos a pagar 5000 euros por el software.

Si un cliente quisiera un proyecto similar al nuestro y su precio a pagar fuera de 5000€ obtendríamos un esfuerzo de:

$$Esfuerzo\ = 5000€ \div\ 2035€\ \frac{sueldo}{mes} trabajador = 2{,}45\ trabajadores$$

Ahora podemos establecer de cuánto tiempo estamos hablando, ya que conocemos el esfuerzo y sabemos que tenemos un solo trabajador:

$$Tiempo\ = 2{,}45\ trabajadores\ \div 1\ \frac{trabajadores}{mes}\ = 2{,}45\ meses$$

Estaríamos hablando de 2 meses y 2 semanas de trabajo para llevar a cabo el proyecto.

No obstante, el precio mínimo para establecer este proyecto sería de 5595 € (coste que hemos calculado arriba con la Ley de Parkinson), con el que cubriremos sueldos y gastos.

Con estos dos datos, vamos a ver en función de los ingresos estimados que podríamos obtener, cuanto tiempo necesitaríamos para poder llegar a cubrir tanto el dinero que el cliente estaría dispuesto a pagar por el software, como el que nosotros realmente necesitaríamos para cubrir todos los gastos del mismo.

Para poder calcular estos datos es necesario entender que el modelo de negocio de la aplicación estará basado en anuncios. Estos anuncios saldrán en determinados momentos del flujo de ejecución del juego. Los anuncios elegidos serán "*Interstitial ads*", concepto que se explica en el siguiente punto con detalle, que consta básicamente en los típicos anuncios que vemos en todas las aplicaciones de hoy en día. Salen de forma natural en el flujo del usuario a través de la aplicación y suelen tener la opción de poder cerrarlos.

Aparte de tener medianamente claro el modelo de negocio con su elección de anuncios, deberemos tener otros conceptos claros como el eCPM. Significa literalmente "coste por cada mil impresiones". Es un valor calculado en función del tipo de anuncio mostrado, del país y la edad del usuario, como aspectos más importantes entre otros muchos. Este valor será el que utilicemos para todas las aproximaciones que vienen a continuación.

Si nos centramos a nuestro caso: anuncios de tipo "*Interstitial ads*", visualizados en su gran mayoría en el Oeste de Europa, veremos lo siguiente:

Ilustración 16 Anuncios tipo intersticiales en el Oeste de Europa - Fuente: https://www.blog.udonis.co/mobile-marketing/mobile-apps/ecpms

En este gráfico podemos observar varias cosas, como podría ser la tendencia del último año del CPM, el *share* de cada tipo de anuncios e incluso el CPM de las distintas compañías capaces de proporcionar publicidad a tu producto.

Para este juego, se ha utilizado Unity Ads. Más adelante en el documento están explicados los motivos de porqué se ha continuado dentro de la plataforma Unity y no se ha optado por ejemplo por los anuncios de Google con AdMob.

Entonces, observando el gráfico se puede ver que el eCPM de Unity Ads con Interstitial Ads y para el Oeste de Europa es de 1.67$, o lo que es igual a 1,38€. Por lo que, si volviéramos al ejercicio que estábamos haciendo, podríamos calcular la cantidad de impresiones que necesitaríamos para poder rentabilizar los 5000€ que el supuesto cliente está dispuesto a pagar. Y a su vez, los 5661,74€ que costaría desarrollar el software, dato calculado según la ley de Parkinson más los gastos de hardware y software, calculados en los componentes de costes.

Bien, si cogiéramos estos números e hiciésemos algunos cálculos se obtendría algo similar a esto:

$x =$ *impresiones necesarias para amortizar los* 5000€ que el supuesto cliente está dispuesto a pagar

$$x = \frac{5000\ (euros\ a\ amortizar)}{1{,}38 \div 1000\ (\text{euros por impresión})} = 3.623.188{,}4\ impresiones$$

$y =$ *impresiones necesarias para amortizar los* 5661,74€ que supondría el desarrollo software según la ley de Parkinson

$$y = \frac{5661{,}74\ (euros\ a\ amortizar)}{1{,}38 \div 1000\ (\text{euros por impresión})} = 4.102.710{,}1\ impresiones$$

En base a esto, necesitaríamos aproximadamente entre tres millones y medio, y cuatro millones de impresiones para conseguir amortizar los costes y empezar a generar beneficio.

Por último, deberíamos coger esos datos y asociarlos a un intervalo de tiempo, para tener claro realmente si el producto puede ser beneficioso o no. Por consiguiente, necesitamos tener un número estimado de impresiones por día. Aquí vuelve a ser difícil tener datos reales, principalmente porque son datos privados que ninguna empresa comparte, pero aun así podemos hacer algún tipo de estimación. Para facilitar un poco la lectura del estudio, se va a

hacer de forma lineal, donde se va a suponer que el número de impresiones es constante en el tiempo, desde el primer día al último de los intervalos de tiempo. No obstante, en un caso real la gráfica tendría altibajos constantes.

Si tenemos un número de jugadores diarios de 3500 personas, y de cada uno de esos jugadores ven una media de 5 anuncios por día, obtendríamos lo siguiente:

$$impresiones/día = 3500 \text{ jugadores} \times 5 \text{ anuncios} = 17.500 \ impresiones/\text{día}$$

Con este último dato, ya tenemos toda la información necesaria para poder hacer una estimación en tiempo para llegar a amortizar el coste o la inversión.

Total impresiones/intervalo tiempo	Días	Semanas	Meses	Años
5000€ (precio de ejemplo)	207,04	29,57	6,68	0,57
5661,74€ (coste ley de Parkinson + costes)	234,44	33,49	7,56	0,64

Tabla 57 Impresiones en función de intervalos de tiempo - Fuente: elaboración propia

Finalmente, y con todas las incógnitas resueltas, podemos concluir con que se necesitarán entre unos 200 y 230 días para paliar costes y que, a partir de ahí, tanto como si un cliente comprase el software como si nosotros decidiéramos autofinanciarlo, comenzaríamos a generar beneficios.

5.2.2.4 Comparativa y discusión de los valores obtenidos

En resumen, el desglose de **los componentes de coste** proporciona el costo más detallado para cada uno de los gastos que pueden estar involucrados en el desarrollo de la aplicación. Este método es muy eficaz cuando se conocen todos los recursos que se utilizarán y sus costos. Cuanta más información tengamos, mejor se ajustarán los costes totales.

Después, aplicando **la ley de Parkinson** se obtiene el precio de coste de la preparación completa de la aplicación y, por tanto, el precio mínimo por el que deberíamos venderla. Esto resulta de los datos del cálculo de los componentes de coste.

Por otro lado, Pricing to Win, debido a las condiciones que impone el proyecto, no ayuda realmente a saber el costo real de la aplicación. Aun así, se han estimado tarifas con las que se calcula el tiempo necesario para recuperar la inversión. Lo cual ayuda a tener una versión de negocio más clara sobre el proyecto.

En resumen, la mejor manera de estimar el costo de un tipo de proyecto como este es estimar los componentes de coste, siempre y cuando, se tenga una cantidad significativa de información para agregar valor a los datos pertinentes.

5.2.3 MODELO DE NEGOCIO

Hoy en día, hay dos grandes vertientes de cómo generar beneficios a través de una aplicación móvil. Una de ellas, podríamos denominarla como premium, donde los usuarios deberán pagar un coste previamente puesto por el vendedor de la aplicación. Es un tipo de mercado, utilizado generalmente por las grandes empresas desarrolladoras, ya que generalmente los usuarios no gastan dinero en aplicaciones que no sean de fuentes de confianza (google, 2021).

Por otro lado, tenemos las aplicaciones freemium. Que consisten en aplicaciones a coste cero, donde el desarrollador genera beneficios de forma pasiva mediante otros medios. Aquí hay varias formas de acabar monetizando el juego y conseguir beneficios del mismo.

Esta es una lista de modelos de negocio basados en las aplicaciones freemium:

- Anuncios en la aplicación

La publicidad en la aplicación es la forma más popular de ganar dinero con su aplicación. Todos hemos tratado en alguna ocasión con anuncios en una aplicación, ya sea tener que ver un anuncio corto o hacer clic en un anuncio para poder conseguir alguna recompensa o desbloqueable. Un modelo completo donde los usuarios pueden utilizar la aplicación de forma gratuita y los desarrolladores ganan dinero con la publicidad.

Hay una multitud de anuncios diferentes, los cuales varían según el tipo de aplicación. Para que sean más efectivos, se eligen según el género de la aplicación, el subgénero, rango de edades de los usuarios, países donde se usa y un largo etcétera de condicionales.

- Monetización de datos

Si la publicidad no se ajusta a los requisitos de tu aplicación, otra opción podría ser la monetización de datos, especialmente si tienes una gran base de usuarios. La monetización de datos consiste literalmente en monetizar los datos de tus usuarios. Cada vez que un usuario interactúa con su aplicación, se recopila información. Esta se puede anonimizar y vender a un recopilador de datos, como anunciantes o comerciantes.

La monetización de datos se utiliza en muchas industrias y, en la industria de software es una forma de ganar dinero con su aplicación sin molestar a los usuarios, ni interrumpir su interfaz de usuario con un anuncio molesto. Muchas empresas buscan estos datos, debido a que pueden proporcionar información valiosa sobre el comportamiento del consumidor. Aunque este tipo de modelo negocio es peligroso o incluso poco ético siempre que los usuarios no estén debidamente informados.

- Compras en la aplicación

Este modelo es perfecto para muchos juegos móviles. Las compras dentro de la aplicación son compras que generalmente desbloquean una nueva función o le dan al usuario alguna ventaja que antes no tenía. Estas compras son una excelente manera de monetizar un juego, especialmente cuando en esa tienda de objetos hay algunos que son necesarios o importantes, para que el usuario avance en el recorrido del juego.

Es una excelente manera de ganar dinero e incentivar a los usuarios a jugar más, pero obviamente no se puede ajustar a cualquier juego. Por suerte, como ya hemos visto, hay más alternativas (adsbalance, 2021).

Una vez que se han visto los tipos de modelo de negocio que hay, se van a asociar al proyecto en cuestión.

En este caso, el modelo de negocio elegido ha sido basado en ingresos pasivos. Es decir, un modelo **freemium,** donde la aplicación tendrá un coste de descarga de cero. Sin embargo, habrá anuncios embebidos dentro de la misma. Estos anuncios irán generando ingresos pasivos de una forma continua. Vamos a ver los distintos tipos de anuncios que nos podemos encontrar en una aplicación, para poder asociarlos después con la elegida para el proyecto.

Se pueden encontrar los siguientes tipos de anuncios:

- Anuncios de banner

Suelen ser una imagen que ocupa un lugar en pantalla. Por lo general, se sitúan en la parte superior o inferior de la pantalla mientras la aplicación esté en ejecución.

- Anuncios intersticiales

Los anuncios intersticiales son anuncios que ocupan la pantalla completa. Aparecen en pausas naturales en el flujo de una aplicación; por lo general, ofrecen la opción de salir y omitir. Estos son personalizables y garantizan una interrupción mínima en la experiencia del usuario y, al mismo tiempo, maximizan los ingresos de la aplicación.

- Anuncios de video recompensados

Los anuncios de video recompensados ofrecen a los usuarios recompensas en la aplicación por ver un anuncio. Este tipo de anuncio es una ventaja para todas las partes involucradas, usuario, desarrollador y anunciante, porque los usuarios obtienen contenido gratuito en la aplicación, los desarrolladores ganan dinero con el anuncio y los anunciantes disfrutan de una mayor visibilidad y altas tasas de clics.

- Anuncios reproducibles

En pocas palabras, anuncios que se pueden reproducir. Este tipo de anuncio permite a los usuarios sentir como si hubiesen pagado por un juego con un tiempo límite de un minuto, ofreciéndoles una experiencia agradable y la oportunidad de probar el juego. Es decir, es como probar una demo que aparece en forma de anuncio en el flujo habitual de una aplicación (adsbalance, 2021).

Una vez explicados los distintos modelos de negocio y, posteriormente, las distintas formas de ganar dinero con el modelo *freemium* y, los distintos tipos de anuncios dentro del tipo de anuncios en la aplicación, se va a aplicar al proyecto que se está tratando.

Como ya hemos dicho previamente, se ha elegido un tipo de modelo de negocio *freemium*, con anuncios dentro de la aplicación como subgénero. Estos anuncios serán de tipo **intersticiales**. Irán acoplados dentro de la misma aplicación, y aparecerán en función de los eventos que vayan transcurriendo dentro del juego.

Se han elegido este tipo de anuncios, debido a que, aunque se pueden acoplar perfectamente el resto de tipos de anuncio al software a desarrollar, cualquiera de los otros empañaría la experiencia visual o jugable del mismo. Los banners le restarían importancia visual al menú y su *parallax* y, tanto los anuncios de video recompensados como los anuncios reproducibles, cambiarían el flujo del juego. Ya que los videos recompensados normalmente generan ventaja o hacen esperar al usuario para poder seguir jugando y, los anuncios reproducibles, normalmente no se pueden omitir. Así que, sin duda para conseguir un juego que sea una mezcla entre el estilo "*Flappy Bird*", en el que lo único importa es jugar y, un apartado visual algo más trabajado, los anuncios intersticiales son la mejor elección.

Para finalizar el apartado y, como pequeño prólogo de lo que se verá en el apartado de software utilizado, se ha utilizado Unity Ads para poder poner en práctica todo este modelo de negocio previamente anunciado. Simplemente, se ha elegido por las ventajas y la facilidad para integrarlo correctamente, además de la "ayuda" que proporciona a proyectos pequeños.

5.2.4 NICHO DE MERCADO (PÚBLICO OBJETIVO)

Se va a empezar hablando de cifras, y no hay mejor manera para entender rápidamente la magnitud del sector de los videojuegos que con una gráfica. Estos datos son relativos a los videojuegos para dispositivos móviles.

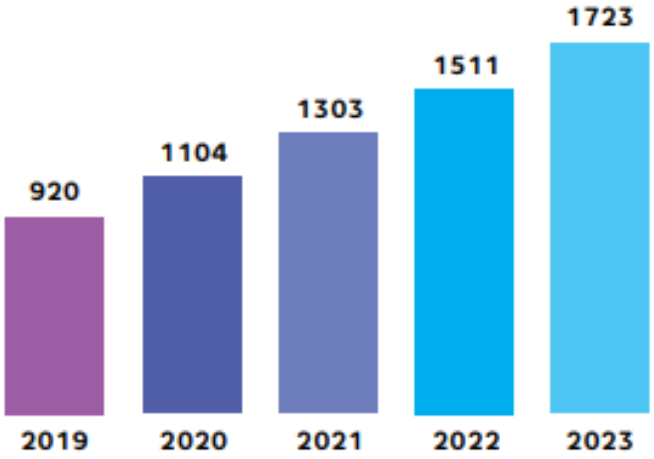

Ilustración 17 Ingresos en millones del sector de los videojuegos en España, dividido por años - Fuente: (DEV - Asociación Española de Empresas Productoras, 2020)

Como se puede observar en la ilustración 17, existe un nicho de mercado en crecimiento, en gran parte, por la continua expansión del mercado de dispositivos móviles.

Si desglosáramos esta información por rangos de edad y sexo, obtendríamos lo siguiente:

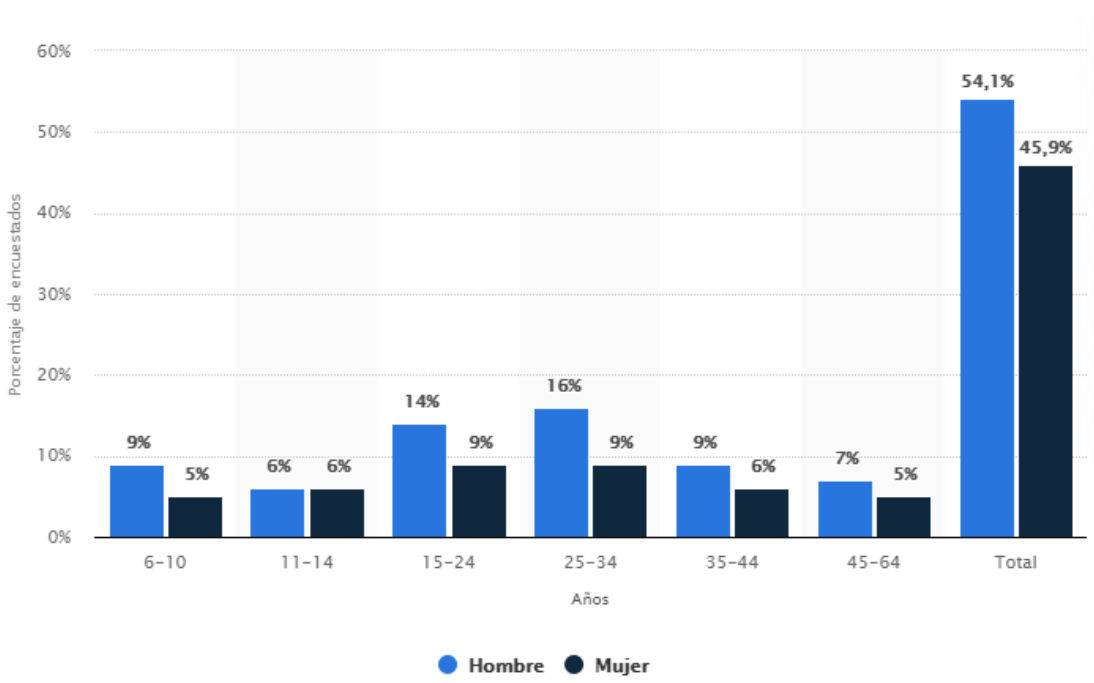

Ilustración 18 Estadísticas según el sexo de los jugadores - Fuente: https://dev.org.es/images/stories/docs/libro%20blanco%20del%20desarrollo%20espanol%20de%20videojuegos%202020.pdf

Donde se puede observar que la mayoría de jugadores siguen siendo hombres, aunque con poco margen, y que el rango de edades con mayor número de jugadores se sitúa entre los 25 y los 34 años. Seguido de cerca por el rango colocado de los 14 a los 24 años.

Toda esta información es importante debido a que el CPM mencionado en anteriores apartados, depende en gran parte del país donde se produce la visualización del anuncio, pero también de la edad de este.

Por otro lado, Roboto Runner busca jugadores que tengan cinco minutos para jugar, que no quieran profundizar, que quieran algo satisfactorio desde el primer minuto, con una curva de aprendizaje muy corta. En resumen, un estilo de jugadores muy casuales.

Desde que el mercado de los dispositivos móviles apareciese, el número de jugadores habituales al género ha ido creciendo de forma exponencial, reviviendo poco a poco épocas de antaño.

Por tanto, no es ninguna novedad que este tipo de género cuente con un número de jugadores increíblemente alto. De hecho, la combinación de dispositivo móvil más este estilo de videojuego totalmente casual genera un nicho de mercado desproporcionado.

Para poder entender este gran nicho de descarga, se puede observar la siguiente tabla:

NÚMERO DE DESCARGAS	
Temple Run	557.369.438
Temple Run 2	634.143.230
Pac – Man 256	+10.000.000
Spider-Man Unlimited	+30.000.000
Jetpack Joyride	260.241.755

Tabla 58 Número de descargas de algunos juegos Android - Fuente: https://www.androidrank.org/

En la tabla, se puede ver el número de descargas de los juegos comentados previamente en la introducción. Aunque estos son simplemente algunos ejemplos de la enorme cantidad de descargas que este tipo de juegos pueden llegar a tener.

Con toda esta información vista anteriormente y algunos datos más como podría ser el tiempo de juego medio por día, existen algunos algoritmos capaces de hacer estimaciones de la cantidad de ingresos que generan estos juegos. Se va a mostrar en una tabla con los mismos juegos de arriba:

	Ingresos/semana	**Ingresos/mes (aprox.)**
Temple Run	$3380 – 2849€	$13520 – 11396€

Temple Run 2	$6142 – 5177€	$24568 – 20708€
Pac – Man 256	$2222 – 1873€	$8888 – 7492€
Jetpack Joyride	$21272 – 17930€	$85088 – 71720€

Tabla 59 Ingresos estimados por semana en dólares y euros - Fuente: https://java8test-dot-reflection-live.appspot.com/#!app:iOS/457446957/null/ratings-and-info/2020-08-04/2020-09-04/false/iPhone/iPhone,iPad/United-Kingdom/all-countries

*Spider-Man Unlimited no aparece debido a que desapareció de la Play Store hace años y, por tanto, no hay datos disponibles.

Para concluir, podríamos afirmar que tenemos un nicho de mercado con un notable crecimiento, pero en el que ya existe un espacio creado. Un lugar amplio en el que intentar complacer a sus habituales. Lugar en el que abundan jugadores que buscan una experiencia accesible, rápida, placentera desde el primer instante. Un nicho altamente poblado, no obstante, con una gran y feroz competencia.

5.3 Herramientas software utilizadas

En este apartado se van a detallar las herramientas utilizadas para llevar a cabo el desarrollo de este punto, el análisis.

Respecto a los primeros apartados, requisitos, riesgos y, componentes de coste; no se ha utilizado ninguna herramienta de software especializada, nada más allá de una calculadora para poder realizar los cálculos.

Sin embargo, para el modelo de negocio y los apartados derivados, se ha utilizado una herramienta para la inserción de publicidad intersticial, y por tanto la monetización del software.

Aquí, hay muchas herramientas y proveedores distintos e incluso hay empresas encargadas de ir saltando de una red publicitaria a otra, para poder maximizar beneficios en función de las políticas actuales de cada herramienta.

A pesar de que hay un sinfín de redes publicitarias distintas, para los juegos desarrollados en Android hay dos mayoritarias que destacan por encima del resto. Estas son Google AdMob y Unity Ads. Vamos a hacer una pequeña comparativa.

Ambas son herramientas capaces de monetizar juegos para Android, pero aún siendo similares, poseen instrumentos de integración distintos, políticas distintas e incluso un CPM (concepto previo), completamente distinto.

AdMob por un lado, presenta un CPM más alto excepto en ocasiones especiales. Tiene más posibilidades, más tipos de anuncios distintos, una comunidad más grande… Aunque no todo

son ventajas. Las campañas son confusas, a veces funcionan muy bien, otras directamente no lo hacen. Consecuencia de que las políticas de Google son variables, y el funcionamiento de los algoritmos no es tan claro como debería. Si la aplicación deja de tener usuarios durante un tiempo es posible que dejes de generar ingresos, o de que entres en una especie de modo oculto en el que casi no generas ganancias.

En el otro costado, está Unity Ads, con un CPM más bajo, pero con una integración sencillísima, con apoyo total a proyectos muy pequeños, sin proyectos que dejan de generar ingresos sin motivo, y con un soporte más cercano que el que ostenta Google (GamePlayDeveloper, 2021).

Toda esta información la podemos ver en la ilustración 16 vista anteriormente.

Con todo, se ha elegido **Unity Ads** como herramienta para monetizar el juego. La explicación es muy simple: el juego está desarrollado en Unity y, este provee de un montón de instrumentos y facilidades para la integración del plugin en el juego. Es cierto que los ingresos son menores y que tiene menos posibilidades, pero ese aspecto no es esencial para este proyecto. Dispone de los tipos de anuncio que se quieren mostrar y, principalmente, no te limita la posibilidad de generar ingresos en caso de que el alcance de tu juego pase por altibajos.

6 Diseño del juego (Game Design Document)

En esta sección, se va a hacer un profundo paseo, por todas las áreas visibles y palpables del videojuego. Se describirán todas las áreas relacionadas con el diseño del juego, desde puntos muy básicos como el estilo y el género del videojuego, hasta la ambientación elegida. Todo esto incluye una gran cantidad de apartados que relatan cosas como: las decisiones sobre la estética final, conceptos visuales, la interfaz del juego, elección de mecánicas, estilo de jugabilidad, tipo de controles e incluso las herramientas software utilizadas.

6.1 Ficha técnica

Pequeño anexo con toda la información del videojuego:

- Título: Roboto Runner
- Plataforma: Dispositivos móviles Android
- Género: Endless Runner
- Audiencia: Todas las edades
- Formato: Apaisado
- Número de jugadores: Un jugador
- Logros: Logros de Google Play Juegos
- Estadísticas: Estadísticas de Google Play Juegos
- Idioma: Inglés

Ilustración 19 Logo Roboto Runner - Fuente: elaboración propia

6.2 Historia

Roboto Runner, es un juego directo, sin explicación explícita de la historia. Empiezas a jugar y directamente apareces en partida, una y otra vez, como si de un bucle se tratase. Bucle donde un pobre robot corre indefinidamente para persistir en el tiempo, teniendo que esquivar a su paso una serie de obstáculos, entre los cuales hay otros robots.

Puesto que no se específica en ningún momento el arco argumental de la historia, cada jugador puede interpretar qué ha pasado para que se haya podido llegar a una situación como la que se desarrolla a lo largo del juego o, cómo podría acabar la misma. Cada usuario podría crear su propio *lore* e intentar descifrar el cómo y el porqué.

Por otro lado, aunque no haya un *lore* explícito, sí que hay un razonamiento y una idea detrás de todo. Es la siguiente:

El dueño de una compañía ha introducido su conciencia en un robot para poder batallar contra otros robots, debido a que estos, se han vuelto malvados. Ser parte de ellos le ayudará a pasar desapercibido, pero desgraciadamente, la inteligencia artificial de estos robots es superior de

lo esperada y se han dado cuenta de que está ocurriendo. Los peligrosos robots planean cerciorarse de nuestro amigo roboto y hacerle pagar las consecuencias.

Para finalizar el apartado historia, se va a lanzar una pregunta: ¿es importante la historia? Un videojuego ha de ser divertido y aunque la historia puede ser algo muy válido, no tiene porqué ser imprescindible. E incluso con una historia marcada, es importante dejar espacio a la imaginación.

6.3 Género

Roboto Runner es un videojuego de estilo 2D, donde tanto el escenario como el fondo y los objetos que aparecen a lo largo de una partida son planos. Las animaciones están hechas con *sprite sheets*, donde cada uno de los fotogramas que componen estos *sprites* están dibujados digitalmente y son de realización propia.

Unity puede simular un entorno 2D en un espacio 3D. Por tanto, es la cámara la que realmente da la sensación de mundo plano sin profundidad, aunque realmente sigue siendo un mundo con tres dimensiones (ancho × altura × profundidad).

En la siguiente ilustración se puede apreciar este entorno 2D emulado en un mundo 3D:

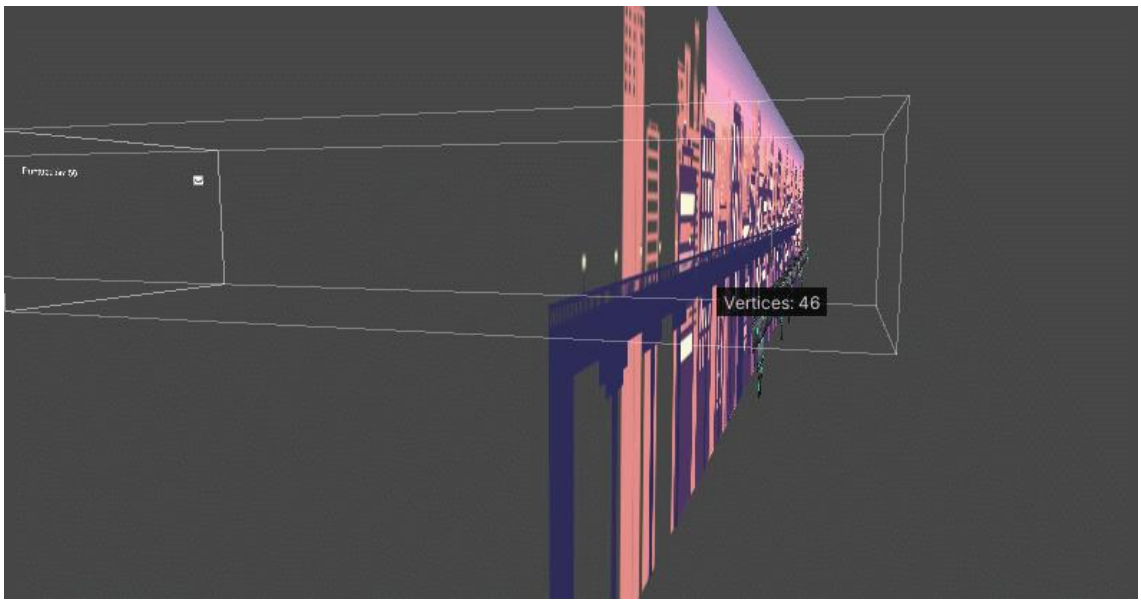

Ilustración 20 Entorno 2D de Unity - Fuente: elaboración propia

6.4 Ambientación

La ambientación del juego está basada en un aspecto retrofuturista. La ciudad que se ve como fondo constante mientras el videojuego transcurre es una ciudad de estilo contemporáneo, mientras que todo el escenario interactivo con el personaje principal tiene un diseño futurista altamente caracterizado por robots.

La paleta de colores está inspirada en el atardecer, usando colores cálidos y suaves. Cada una de las capas del fondo tiene un color más claro u oscuro dependiendo de la posición en la que se encuentre, para así poder darle profundidad a la escena.

Por otro lado, el personaje tiene una paleta de colores fácilmente distinguible con el escenario. Asimismo, la parte de escenario por donde el personaje va saltando e intentando esquivar obstáculos, tiene unos colores cercanos a los del personaje para dar a entender, de una forma rápida y sencilla, que es el camino que el personaje ha de seguir.

6.5 Personaje principal

Un robot es el protagonista del videojuego, el cual está perfectamente diseñado para sortear y saltar obstáculos a su paso.

Este robot va reaccionando a los estímulos que le van apareciendo por pantalla. Para poder gestionarlos tiene a su disposición unas mecánicas.

Posee unos colores que le diferencian del fondo y que, a la vez, le relacionan con la plataforma principal por la cual avanza indefinidamente.

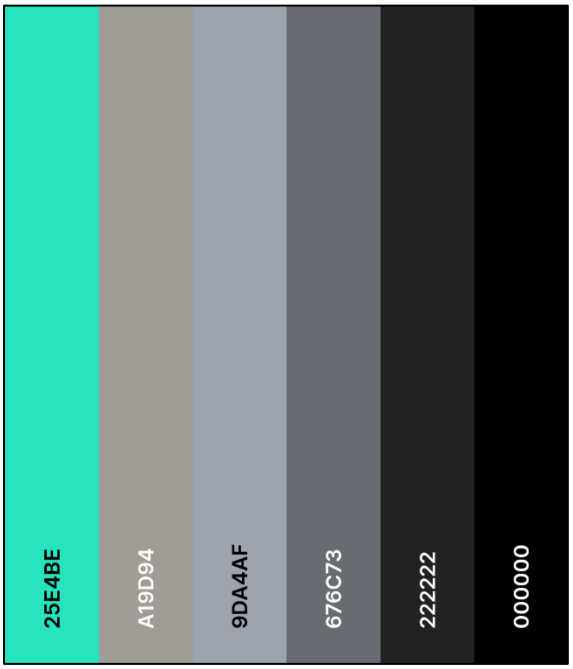

Ilustración 21 Diseño personaje principal - Fuente: elaboración propia

Ilustración 22 Paleta de colores del personaje principal - Fuente: elaboración propia

6.6 Escenario

Hablamos de escenario como elementos que constituyen el entorno que envuelve al personaje y que no son capaces de destruir al personaje. Consta de dos partes independientes tales como el efecto parallax y la plataforma principal de juego.

6.6.1 Fondo – *Parallax*

El fondo está formado por una serie de capas, cada una de ellas forma un *skyline* distinto y, con un color diferente para dar sensación de profundidad. A su vez, cada una de estas capas se mueve a una velocidad distinta. Esto es posible gracias a un script que gestiona la velocidad de movimiento de cada una de estas capas. Todas las capas son ilustraciones de creación propia.

Ilustración 23 Efecto Parallax - Fuente: elaboración propia

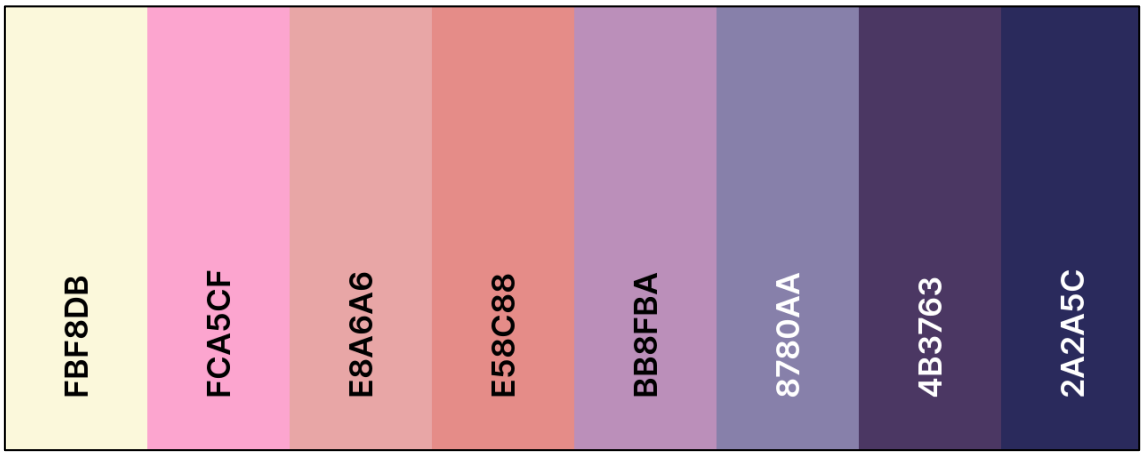

Ilustración 24 Paleta de colores efecto parallax - Fuente: elaboración propia

6.6.2 Plataforma principal

La plataforma principal es el bloque por el que el personaje va corriendo la mayoría del tiempo. Este bloque, al igual que el parallax, se repite de forma indefinida. La paleta de colores es similar a la del personaje para indicar que este debe avanzar por dicha plataforma. Es un *Sprite Sheet* con tres ilustraciones distintas.

Ilustración 25 Plataforma principal - Fuente: elaboración propia

La ilustración 25 muestra lo que hemos comentado. Asimismo, podemos observar que la paleta de colores es la misma que el personaje, la cual se puede ver en la ilustración 22. Esto ya se ha comentado previamente, pero básicamente es para guiar al jugador de forma visual, al tener los mismos colores que el personaje le indica por donde debe ir.

6.7 Enemigos

En este apartado se va a enseñar todos los gameObjects que contiene el videojuego asociados a una caja de colisiones y que por tanto, tienen la capacidad de acabar con el personaje en caso de colisión. Se van a ver los distintos tipos que hay.

6.7.1 Cajas

El primer y más versátil de los distintos enemigos que hay. Se encuentra de distintas formas y se repite bastante en el desarrollo del juego.

Tiene las siguientes características:

- Son objetos pegados a la plataforma principal del videojuego.
- No se mueven al contactar con el personaje ni por sí mismas.
- La partida acaba en caso de que colisiones con alguno de los laterales de esta.
- Solo se pueden esquivar con la mecánica de salto.
- Hay de dos tipos: con pinchos (matan por contacto por arriba) y sin pinchos (solo matan por el contacto con el lateral izquierdo de la caja).

Repasemos los distintos tipos de cajas que aparecen:

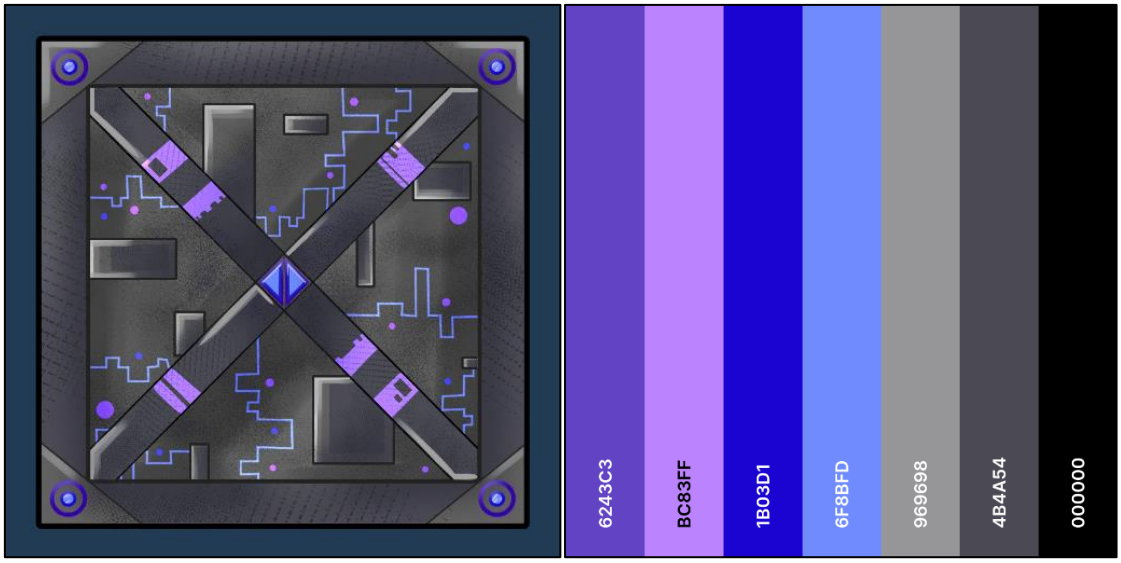

Ilustración 26 Enemigo caja - Fuente: elaboración propia

Ilustración 27 Paleta de colores de enemigo caja - Fuente: elaboración propia

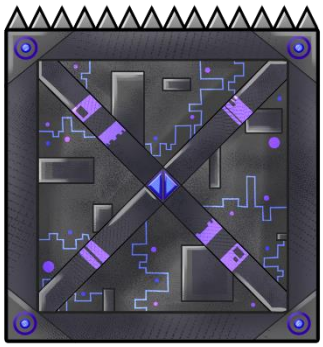

Ilustración 30 Enemigo caja con pinchos - Fuente: elaboración propia

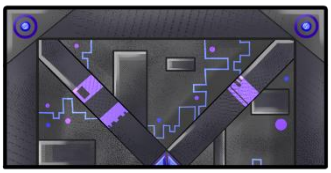

Ilustración 29 Variación 1 caja - Fuente: elaboración propia

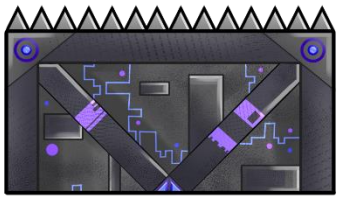

Ilustración 28 Variación 2 caja - Fuente: elaboración propia

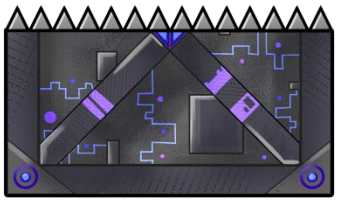

Ilustración 32 Variación 3 caja - Fuente: elaboración propia

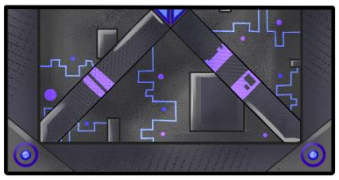

Ilustración 31 Variación 2 caja - Fuente: elaboración propia

6.7.2 Drones

El enemigo intermedio en cuanto a rareza. Presenta varios aspectos distintos, aunque todos tienen la misma funcionalidad

Tiene las siguientes características:

- Levita paralelamente a la plataforma principal del videojuego, pero aparece en diferentes alturas.
- Es inmóvil a la interacción con el personaje.
- Finaliza la partida en caso de colisión independientemente de su tipo.
- Dependiendo de la posición en la que aparezca, se tendrá que esquivar saltando o deslizando.
- Hay varios aspectos de drones.
- Deja una estela a su paso debido a sus motores de propulsión, esta cambia en función del aspecto que tenga.

Ilustración 33 Enemigo Dron 1 - Fuente: elaboración propia

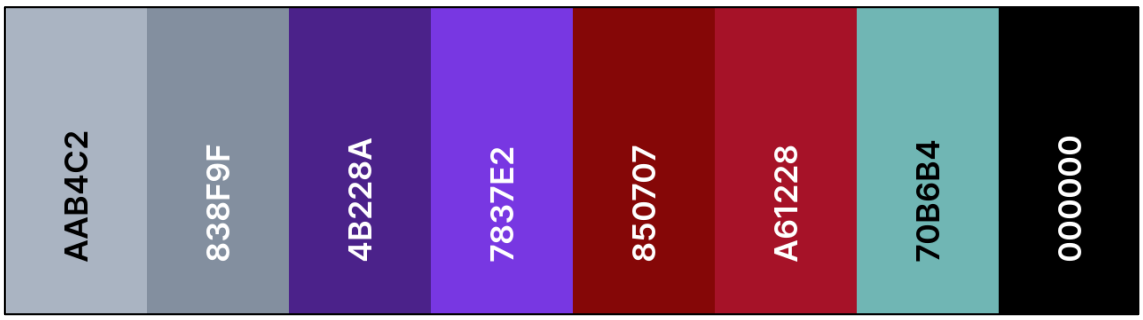

Ilustración 34 Paleta de colores Enemigo Dron 1 - Fuente: elaboración propia

Ilustración 35 Enemigo Dron 2 - Fuente: elaboración propia

Ilustración 36 Paleta de colores Enemigo Dron 2 - Fuente: elaboración propia

6.7.3 Coches

El enemigo menos común. Se encuentra de una única forma y es el más difícil de ver.

Tiene las siguientes características:

- Levita paralelamente a la plataforma principal del videojuego, pero siempre a la misma altura.
- Es móvil cuando el personaje salta sobre él. Además, es un objeto con gravedad y se hunde cuando el robot está encima suyo.
- Finaliza la partida en caso de colisión frontal.
- Solo se pueden esquivar con la mecánica de salto.
- Únicamente hay un tipo de coche.
- Deja una estela a su paso debido a sus motores de propulsión.

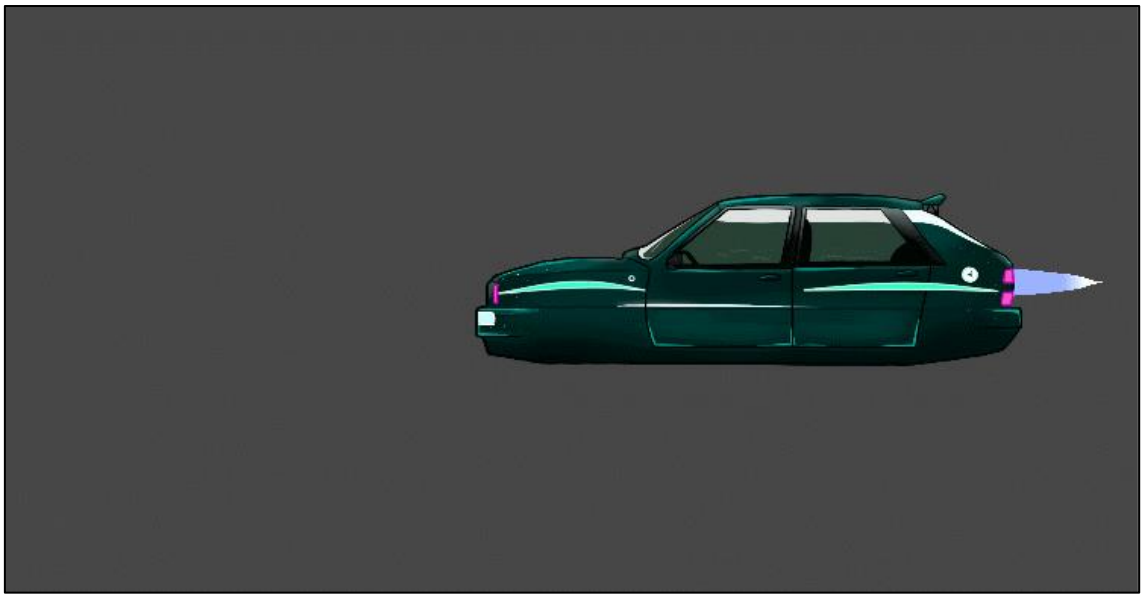

Ilustración 37 Enemigo Coche - Fuente: elaboración propia

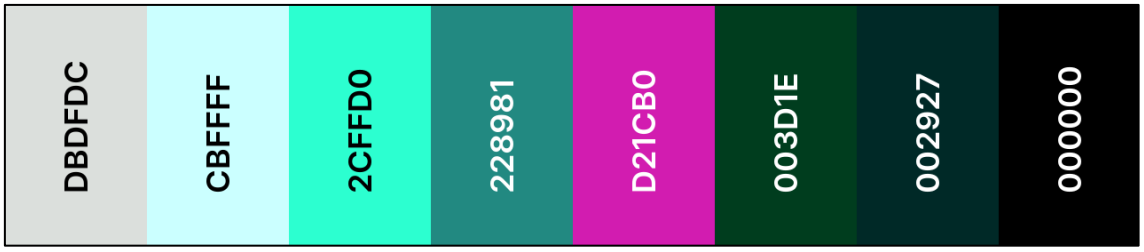

Ilustración 38 Paleta de colores Enemigo Coche - Fuente: elaboración propia

6.8 Jugabilidad

Es un juego sencillo, al estilo *plug and play*, se inicia la partida y no hay más que simplemente jugar. Incluye una curva de aprendizaje prácticamente nula, no obstante, exige una concentración mínima para ejecutar la mecánica necesaria en el momento oportuno. Por esta razón, aunque no haya muchas mecánicas, un jugador debe conocerlas para poder realizar una partida.

Por lo tanto, como con todos los juegos del género, se buscan jugadores casuales que quieran pasárselo bien sin tener que aguantar un recorrido a través de un tutorial o de horas y horas de juego, para poder llegar a aprender o dominar el videojuego en cuestión.

6.9 Mecánicas

Como hemos visto en el apartado de jugabilidad, las mecánicas son sencillas. Vemos tres mecánicas distintas, dos de interacción con el jugador y otra que ocurre constantemente. Todas ellas no tienen dificultad, pero es totalmente necesario que el jugador conozca las tres para poder avanzar en una partida.

Son las siguientes:

6.9.1 Correr

Es la mecánica por defecto. Nada más empezar la partida y de forma continua, el personaje empezara a correr. No hace falta presionar ningún botón ni mover el dispositivo, para que esto ocurra. Aunque no sea una mecánica interactiva, es muy importante calcular las distancias y tener claro el retraso entre el cambio de animaciones.

Así se ve la mecánica formada por su correspondiente *Sprite Sheet*:

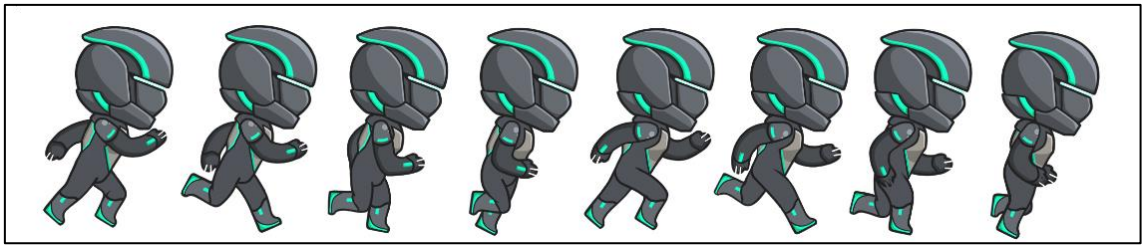

Ilustración 39 Mecánica correr - Fuente: elaboración propia

6.9.2 Saltar

La mecánica más sencilla de usar, pero la más difícil de dominar. Cuando toquemos la pantalla táctil, el personaje saltará una distancia fija, dato importante para tener en cuenta porque la distancia entre enemigos no siempre será la misma.

Mecánica en cuestión compuesta por su correspondiente *Sprite Sheet*:

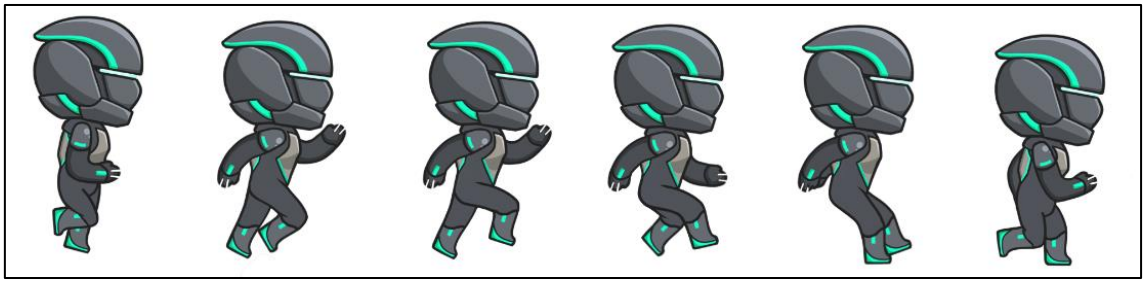

Ilustración 40 Mecánica saltar - Fuente: elaboración propia

6.9.3 Deslizar

Quizá la mecánica que más dificultad conlleva. Al mover el dedo hacia abajo en la pantalla, como si estuvieses navegando por una web, el personaje se tirará al suelo para intentar escabullirse por debajo de los obstáculos que se encuentre.

Este es la animación formada por su *Sprite Sheet*:

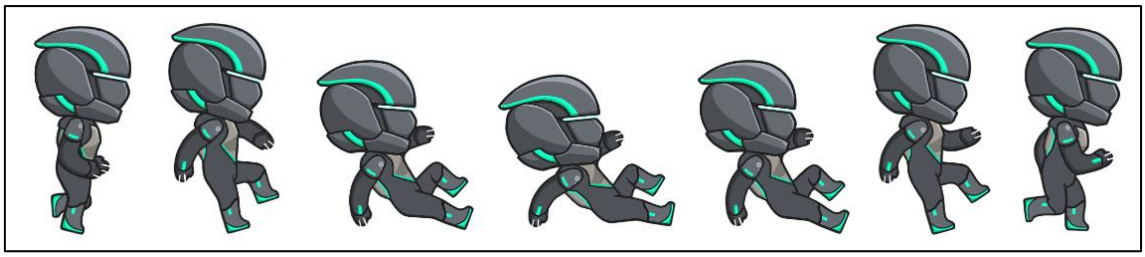

Ilustración 41 Mecánica deslizar - Fuente: elaboración propia

6.10 Controles

Como ya se ha ido mostrando en puntos anteriores de este mismo apartado, el juego está pensado para dispositivos móviles, por lo que todos los controles tendrán que estar pensados para una pantalla táctil.

Por lo tanto, puesto que tenemos tres mecánicas, hay que intentar conseguir que los controles faciliten la jugabilidad y creen una mejor experiencia de juego.

Para lograr esto, se ha actuado de la siguiente forma con cada una de las mecánicas:

- Correr: el personaje por defecto aparecerá corriendo, no será necesario interactuar de ninguna forma con el dispositivo Android. Esto se ha hecho para simplificar la interacción con los controles y dar más importancia a las otras dos mecánicas.
- Saltar: para poder realizar un salto en el transcurso de una partida, será necesario hacer una pulsación táctil en cualquier parte de la pantalla, siempre que el usuario se encuentre en una partida y, excluyendo los botones de pausa.
- Deslizar: al igual que con la mecánica de salto, será necesario interactuar con la pantalla del dispositivo móvil. Para conseguir que el personaje deslice y esquive obstáculos, habrá que deslizar el dedo hacia abajo por la pantalla.

6.11 Escenas

El juego se ha dividido en cuatro escenas, donde cada una de ellas tiene una estética distinta y sus objetos propios, aunque esto lo veremos más adelante. Debido a que estamos en el apartado de diseño, en este punto únicamente se va a ver a adelantar el resultado estético que se ha querido plasmar.

Esta decisión de cuatro escenas ha sido para diferenciar de una forma clara los distintos apartados del juego. Además, con la intención de que tanto la interfaz como la usabilidad de esta sea lo más fácil posible de usar. Si el usuario tuviese problemas para realizar cualquiera de las funcionalidades previstas, supondría un problema serio.

6.11.1 Menú principal

Esta escena consta de tres posibilidades distintas: empezar partida, ver interfaz de controles y salir del juego. Puesto que la interfaz no tiene mucha lógica que manejar, se ha optado por un diseño super básico, aunque llamativo.

De fondo tenemos el mismo efecto parallax que vemos durante el juego, pero con una paleta de colores algo más oscura para poder diferenciar así, los fondos de las distintas escenas. El diseño de los botones es sutil, aunque lo suficientemente llamativo como para indicar al usuario que debe tocar ahí para realizar alguna de las acciones disponibles.

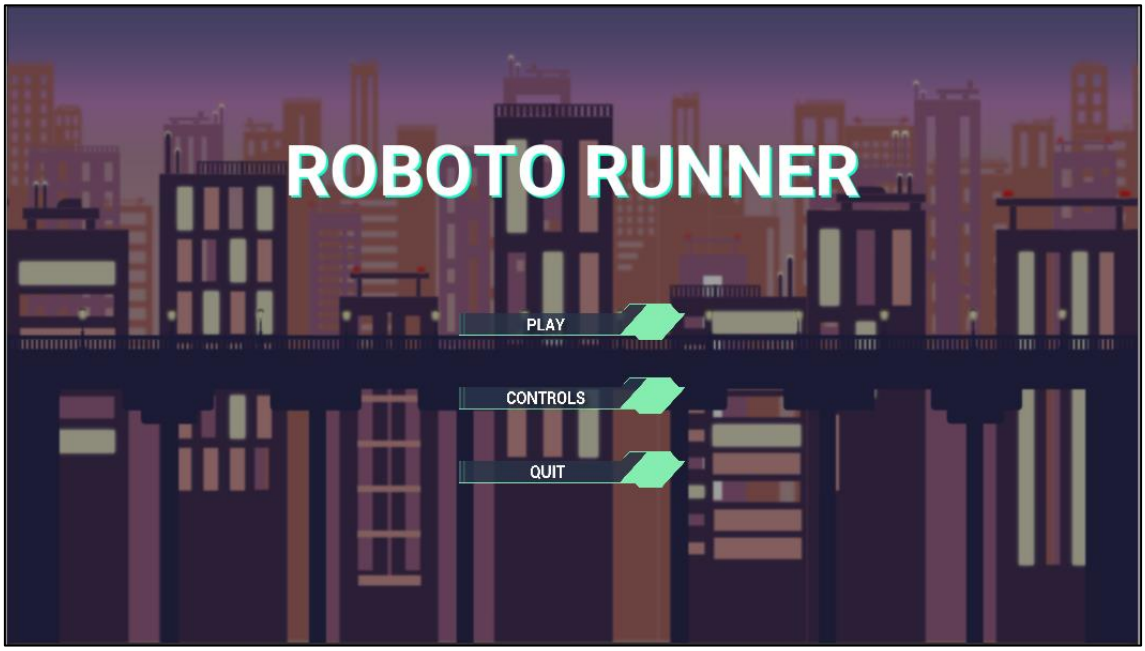

Ilustración 42 Escena menú principal - Fuente: elaboración propia

6.11.2 Menú Pausa

Al igual que con el menú principal, poseerá tres acciones distintas: reanudar partida, ver la misma interfaz de controles y volver al menú principal.

A diferencia del menú principal y de la escena principal, en esta interfaz el fondo permanecerá estático. Del mismo modo, la partida en cuestión se parará y permanecerá en pantalla para el usuario.

Por lo tanto, será la misma partida que tenga el usuario, pero detenida, más los tres botones con las tres acciones plausibles. Con esto se intentará conseguir que el usuario capte fácilmente lo que está ocurriendo al entrar en esta escena.

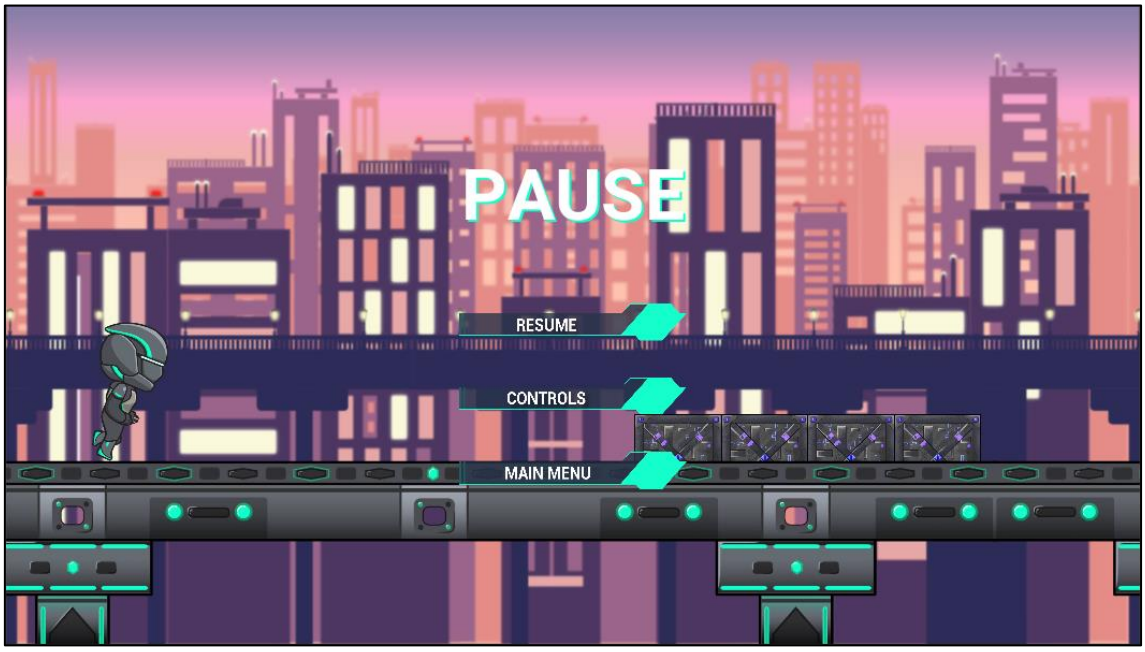

Ilustración 43 Escena menú pausa - Fuente: elaboración propia

6.11.3 Escena Principal

Esta es la escena de juego, por tanto, debe presentar los elementos más importantes en pantalla.

Contendrá tres apartados: escenario de juego, barra de estado y botón de pausa.

- **Escenario de juego**: será toda la parte de la pantalla en la que esté transcurriendo el videojuego, es decir, el tablero de juego.
- **Barra de estado:** parte de la interfaz en la que se verá toda la información relacionada con la partida que el usuario esté realizando. La puntuación de la partida es un ejemplo de la información necesaria en este punto.
- **Botón de pausa:** interruptor que llevará directamente al menú de pausa, el cual permite detener la ejecución de la partida.

En vista de que no todos los apartados tienen la misma importancia, la interfaz se debe definir de forma que deje claro que elementos tienen prioridad sobre el resto.

El escenario o tablero de juego debe ser claramente el aspecto más importante de la interfaz, así que, debe de tener prioridad sobre los otros dos. El segundo más importante tiene que ser

la barra de estado, con lo que tiene que ser visible y legible en todo momento. Por último, tenemos el botón de pausa, el cual ha de estar también en todo instante en la interfaz, pero no debe ser un impedimento visual en ningún momento.

Ilustración 44 Escena principal - Fuente: elaboración propia

6.11.4 Controles

Esta escena es la más sencilla de todas las vistas. Únicamente se deberá mostrar una imagen que sea capaz de explicar de una forma clara y sencilla, cuáles son los controles para realizar las mecánicas del juego.

Por ello, esta interfaz solo tendrá una explicación visual de cómo realizar las mecánicas y un botón para volver al menú de origen.

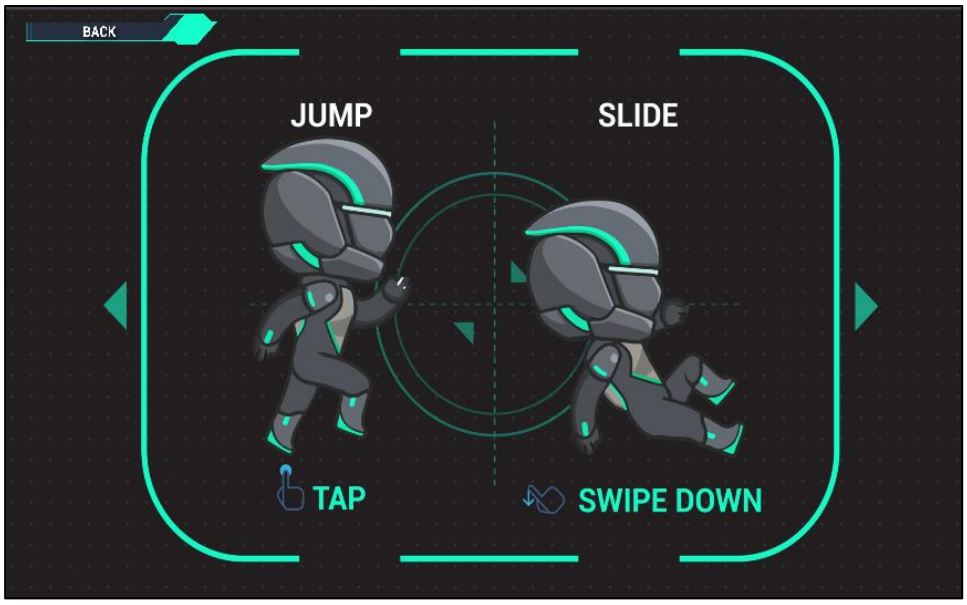

Ilustración 45 Escena controles - Fuente: elaboración propia

6.12 Herramientas software utilizadas

En este punto, se van a explicar las herramientas software utilizadas para el diseño del videojuego. Aunque todas las herramientas hayan intervenido en el diseño del juego, cada una de ellas se ha utilizado para un apartado en concreto.

Como se ha visto con anterioridad en los componentes de coste, Unity posee una licencia de uso gratuita, mientras que Procreate y Adobe Illustrator son herramientas de pago.

A continuación, se explican con más detalle las citadas herramientas:

- **Unity:** se ha empleado para algunos pequeños detalles como el Trail Renderer, herramienta que podemos ver en el diseño de algunos objetos como los drones. Más adelante, existe un apartado que expande esta información sobre el programa Unity.
- **Procreate**: es una herramienta de diseño para iOS. Fue lanzada al mercado en 2011 por la empresa australiana Savage interactive. Presenta multitud de herramientas y una buena gestión de capas, algo imprescindible en el diseño gráfico (Llasera, 2020). Este software, se ha utilizado para el diseño de todos los objetos del juego, desde el personaje principal, sus mecánicas e incluso todos los enemigos que hemos visto previamente.
- **Adobe illustrator**: es un software dedicado al dibujo vectorial y al diseño de elementos gráficos, pudiendo ser usado en casi cualquier ámbito de diseño. Pertenece a la familia de productos Adobe, entre los que podemos encontrar también el famoso PhotoShop. Respecto al juego en cuestión, se ha utilizado para el diseño de todos los fondos que se ven en el proyecto, tanto los del menú principal como los de la escena principal (CreativosOnline, 2014).

7 Desarrollo

En este capítulo se describirá detalladamente el proceso realizado para conseguir plasmar todo lo documentado en el GDD, a su vez, se detallará cómo se han implementado todos los requisitos. Es un apartado trascendental, puesto que, sin un correcto desarrollo, no habrá un buen producto final.

Este capítulo va a hacer un amplio recorrido sobre todas las decisiones relacionas con la implementación del software requerido para llevar a cabo el proyecto. Se empezará con la introducción, donde se verá el motor de desarrollo elegido para el desarrollo del videojuego. Asimismo, se justificará el porqué de esta elección y, alguna pequeña comparativa respecto a sus rivales. A diferencia de con el resto de apartados, se va a dar esta información al comienzo del mismo, ya que, se va a hablar reiteradamente del motor utilizado. Además, se detallará la implementación de algunas de las funcionalidades más importantes, como las mecánicas.

7.1 Motor de Desarrollo

Un motor de desarrollo es un sistema diseñado específicamente para la creación de videojuegos que reúne un conjunto de aplicaciones necesarias para el desarrollo (Carrasco, 2018). Su función principal es proporcionar los medios necesarios para que el juego renderice los modelos y animaciones que componen el videojuego. Habitualmente, el motor suele contener un entorno de desarrollo compuesto por diversas herramientas que facilitan el trabajo de los desarrolladores, como un motor de físicas o un motor de colisiones.

Es necesario comenzar este punto diciendo que no se ha desarrollado un motor de desarrollo propio debido a que es un proyecto individual. Por lo tanto, se ha buscado un motor que cumpla las necesidades requeridas. Puesto que el videojuego está diseñado para ser en dos dimensiones e ideado para correr sobre dispositivos móviles, no hay demasiadas alternativas en el mercado que cumplan estos requisitos.

Actualmente en el sector, a nivel comercial hay dos grandes opciones: Unity y Unreal Engine.

Ambos son motores gráficos realmente potentes y capaces de llegar incluso a soportar el desarrollo de juegos AAA. Un par de ejemplos podrían ser: el Kingdom Hearts 3 desarrollado sobre Unreal Engine o el Deus Ex, desarrollado sobre Unity. Uno y otro, poseen multitud de herramientas y facilidades para el desarrollo, pero cada uno tiene cualidades únicas que hace que dependiendo del caso tengas que elegir uno u otro. Sumado a esto, hay ciertos valores predeterminados en cada uno de estos motores y desarrollar sobre uno u otro, puede hacer que un juego presente cierto estilo predefinido.

Conociendo que cada motor tiene sus propiedades únicas, sabiendo cuales son las necesidades que requiere el proyecto y sin entrar en comparativas detalladas entre ambos motores, ya podemos decir que **Unity** es el motor grafico elegido.

¿Por qué Unity? Mayoritariamente porque ofrece más ventajas y mas facilidades para el desarrollo de juegos 2D. El entorno es más fácil, la curva de aprendizaje es algo menor y las condiciones de uso son buenas. Además, Unity tiene gran cantidad de componentes predefinidos para la creación de juegos en 2D.

7.2 Unity

Bien, ahora vamos a ver una definición un poco más extensa de Unity, para poder comprender mejor la explicación del desarrollo. Como se ha comentado previamente, al contrario de con el resto de apartados del documento, se va a comentar una de las herramientas software utilizadas al principio del apartado. Esto es debido a que es necesario comprender las bases fundamentales del motor para poder comprender el desarrollo de este.

Unity es un motor de videojuego multiplataforma creado por Unity Technologies .Unity está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS, Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataforma (moddb, 2021).

Tiene las siguientes características:

- Utiliza OpenGL (en Windows, Mac y Linux), Direct3D (solo en Windows), OpenGL ES (en Android y iOS), e interfaces propietarias (Wii).
- Permitir el desarrollo de videojuegos en Windows, Mac, Linux, Xbox 360, PlayStation 3 y Vita, Wii, Wii U, iPad, Android, iPhone, etc.
- Dada a su amplia compatibilidad, Unity permite el uso simultáneo con software de modelado 3D (como Blender) y software de gráficos (como Adobe Photoshop u otro software). Aunque esto son solo dos ejemplos, posee una integración directa con gran cantidad de software útil para el desarrollo de un videojuego.
- Desarrollo "sencillo e intuitivo". Obviamente, esto no es fácil ni intuitivo. Puede llevar mucho tiempo desarrollar un proyecto completo, pero considerando la complejidad de los medios, Unity puede considerarse como una de las formas más factibles de desarrollar un videojuego.
- Licencia: gratuita y de pago. Las limitaciones de la versión gratuita no entorpecen el desarrollo libre, pero sí impone algunas restricciones en algunas funcionalidades como el uso de partículas, el uso de texturas y menos elementos predeterminados ... Excepto la pantalla de bienvenida que no se puede borrar al iniciar el juego (Uniovi, 2015).

7.3 Conceptos previos Unity

Con una definición genérica de qué es Unity, se puede profundizar y comenzar a explicar a grandes rasgos cómo es el desarrollo de proyectos en este motor de desarrollo.

Veamos un pequeño libro de definiciones de conceptos de Unity, necesarios para asimilar las explicaciones posteriores:

- Escena: contienen los objetos del juego. Pueden ser usadas para crear un menú principal o niveles individuales entre otros elementos. Piense en cada archivo de escena, como un nivel único. En cada escena, usted va a colocar su ambiente, obstáculos, y decoraciones, el diseño esencial y la construcción de su juego en pedazos.

  En la ilustración 46, se puede apreciar la escena PauseMenu con cada uno de sus distintos *GameObject*:

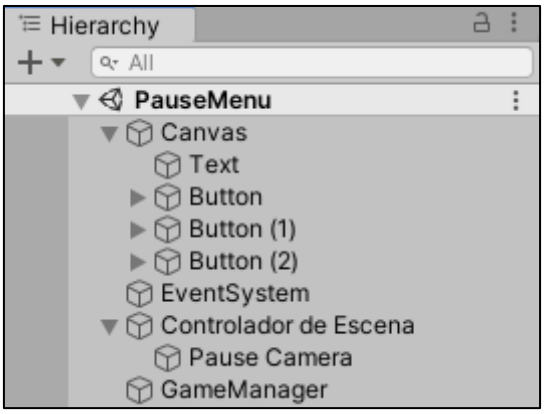

Ilustración 46 GameObjects ejemplo Escena - Fuente: elaboración propia

- GameObject: son objetos fundamentales en Unity que representan personajes, cámara, el escenario, etc. Estos no logran nada por sí mismos, pero funcionan como contenedores para Components, que implementan la verdadera funcionalidad. Por ejemplo, un objeto Luz se crea al adjuntar un componente Luz a un *GameObject*.

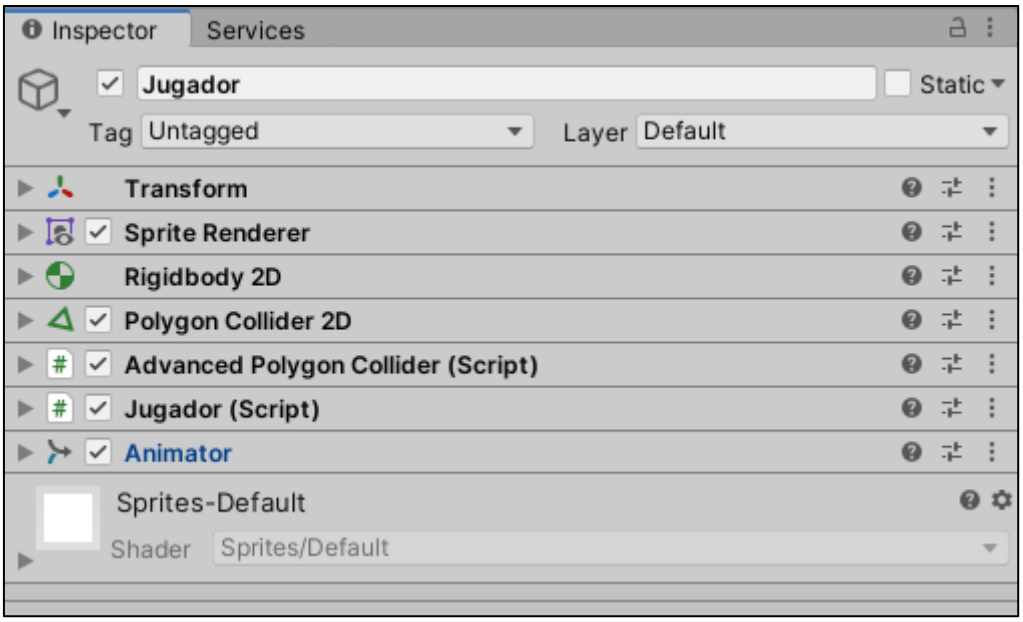

Ilustración 47 Ejemplo GameObject - Fuente: elaboración propia

Como se puede ver en la ilustración 47, Jugador es un *GameObject* compuesto por una serie de componentes que le proporcionan las funcionalidades requeridas.

- Componente: Los componentes son las partes que componen los objetos del juego. Cada uno de los apartados que aparecen para un objeto en el Inspector, se trata de un componente. Estos, generan las funcionalidades requeridas a cada uno de los *GameObject*.

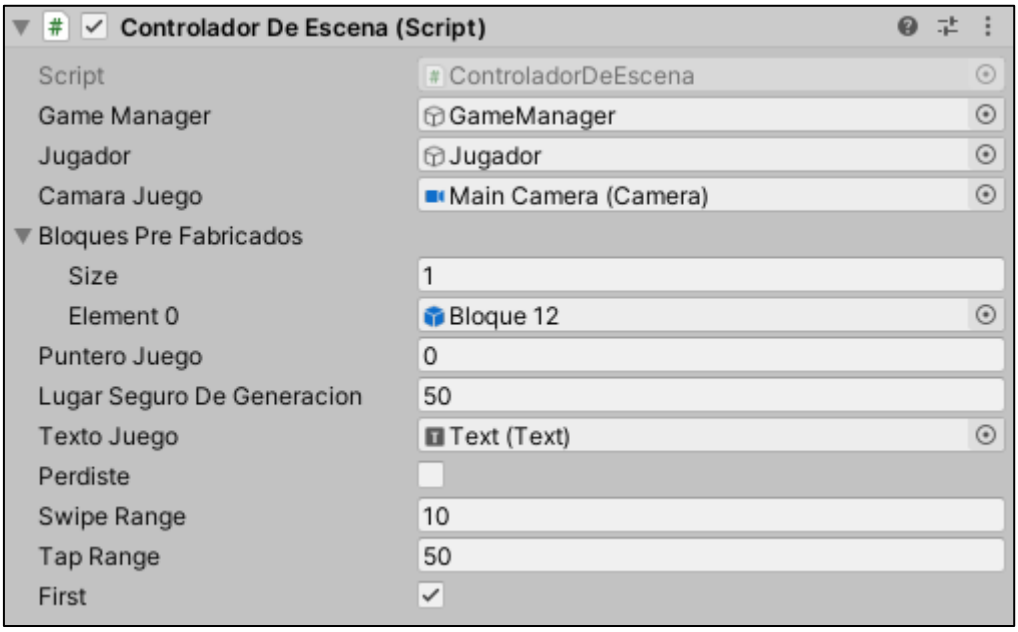

Ilustración 48 Ejemplo componente - Fuente: elaboración propia

En la ilustración 48 se puede apreciar un componente asociado a un script propio llamado Controlador De Escena. Como se puede ver tiene una gran variedad de valores para configurar desde el inspector de Unity .

Unity está basado en la división por escenas. Cada una de estas escenas puede tener diversas concepciones, como podría ser cada nivel de un videojuego o por ejemplo el menú de pausa. Puede haber varias escenas cargadas a la vez, pero sólo puede haber una activa.

Cada una de estas escenas está formada por una multitud de *GameObjects* distintos. Asimismo, todos y cada uno de los *GameObjects*, están formados por multitud de componentes distintos, cada uno de ellos con distintas propiedades y características (Unity, 2021).

Con toda la información necesaria vista, se van a explicar las decisiones tomadas para el desarrollo del proyecto en cuestión.

7.4 Gestión de Escenas

Como hemos visto previamente, las escenas en Unity tienen múltiples interpretaciones posibles y todas de ellas válidas. En este caso, se ha escogido la opción de que cada uno de los distintos estados que compondrían el juego se corresponda con una escena distinta con sus *GameObjects* únicos y a su vez, sus correspondientes componentes.

Algunos de los estados podrían haberse hecho con interfaces que se mostrasen u ocultasen, pero este comportamiento quedó descartado debido a que, aunque la implementación quizá fuese un poco más sencilla, dificultaba su entendimiento.

Por lo tanto, cada uno de los distintos estados que se gestionaría mediante una pila de estados, va directamente asociado a una escena de Unity.

Estas escenas se comunican entre ellas siguiendo un flujo concreto. A continuación, se puede ver el diagrama de flujo que relaciona las escenas:

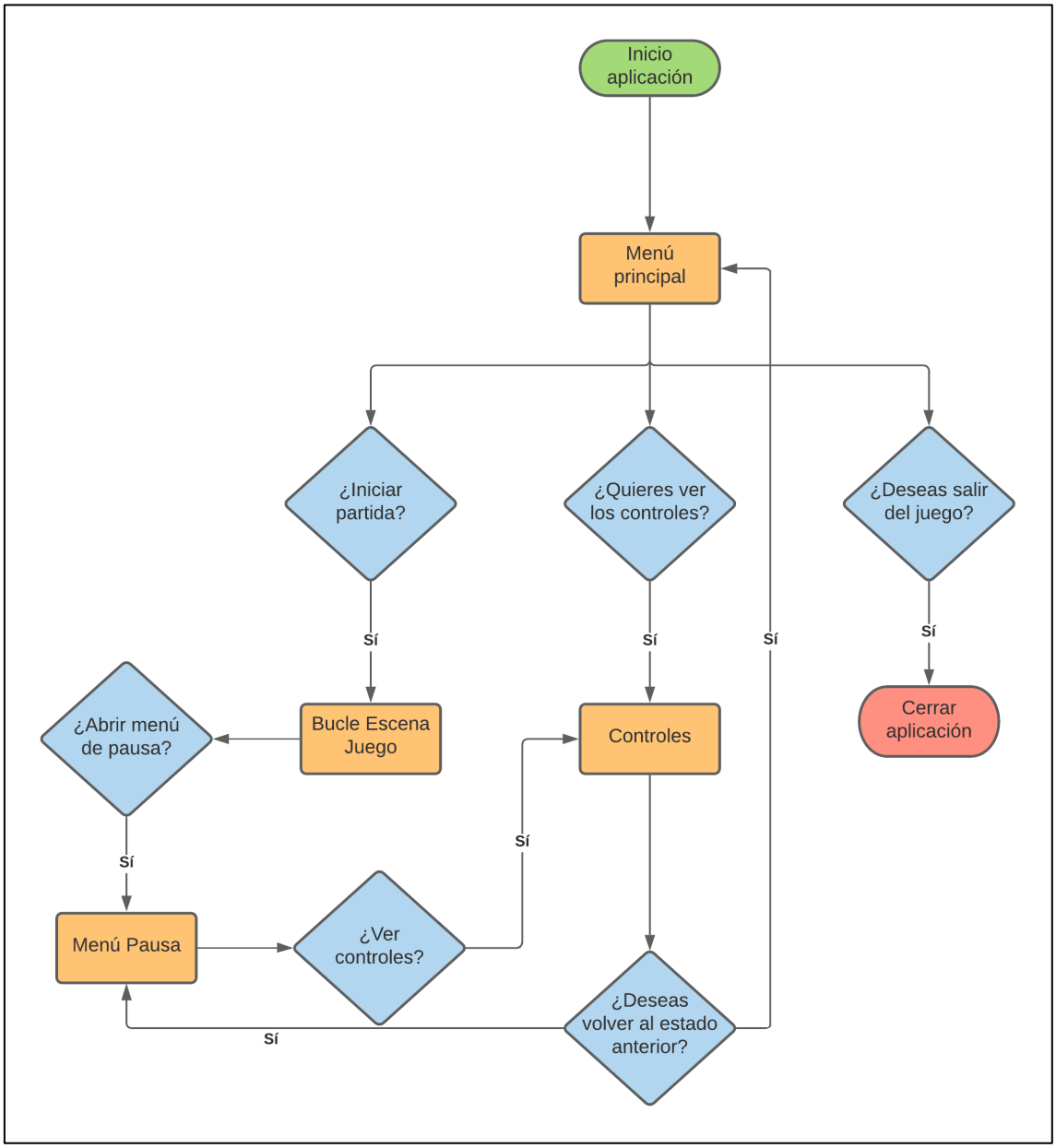

Ilustración 49 Diagrama de flujo escenas simplificado - Fuente: elaboración propia

Una vez aclarada la disposición de escenas, vamos a ver cada una de ellas de forma independiente.

7.4.1 Escena Menú principal

Es una escena sencilla que contiene toda la lógica de botones para poder iniciar partida, mostrar controles o salir del juego.

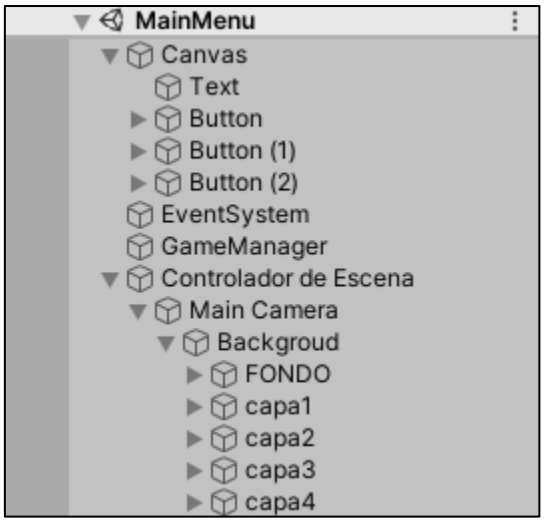

Ilustración 50 GameObjects escena menú principal

Contiene todos los *GameObjects* que se aprecian en la ilustración 50, donde se puede ver un canvas con toda la lógica de botones para la transición entre escenas. Cada uno de estos botones tiene los siguientes componentes:

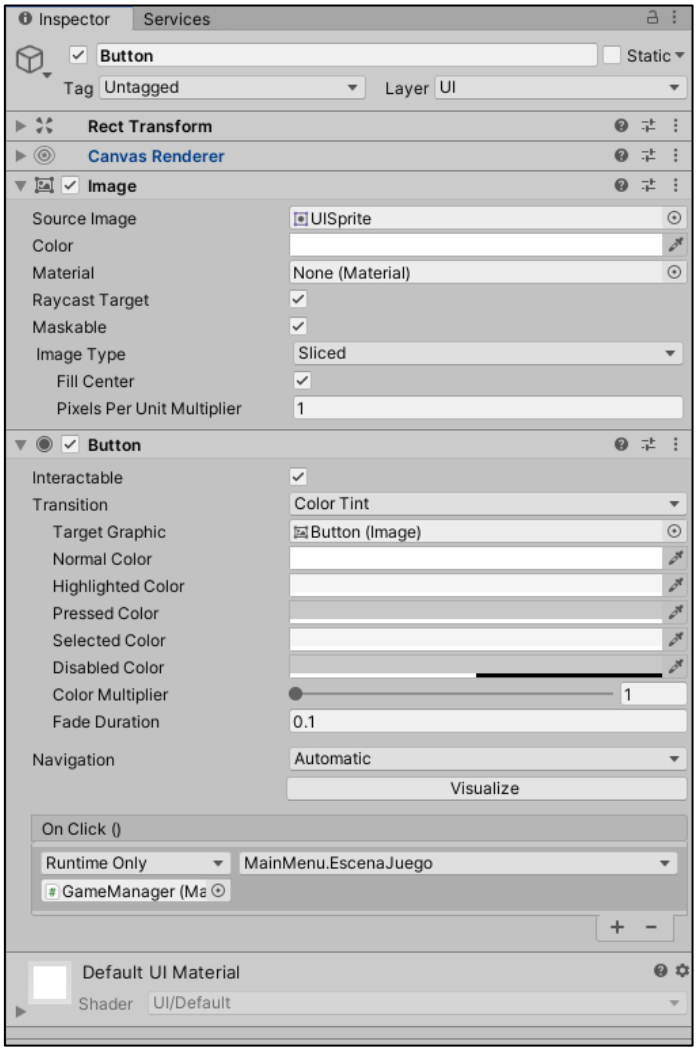

Ilustración 51 Componentes gameObject botón - Fuente: elaboración propia

Aquí, simplemente hay que destacar el componente Button (nativo de Unity), el cual tiene asociado un evento On Click, que sirve para desencadenar la llamada a una función al hacer una pulsación táctil. Se utiliza esta lógica para llamar a un método que carga la escena correspondiente. Por ejemplo:

```
public void EscenaJuego()
{
    SceneManager.LoadScene("InGameScene");
}
```

Igualmente, se puede apreciar como cada una de las capas que componen el fondo en movimiento visto en el GDD, contienen el script auxiliar Parallax. Asignándole un valor de desplazamiento desde la sección de inspector de Unity, conseguiremos el efecto en movimiento visto anteriormente. En el caso de la ilustración que hay a continuación sería el valor 1.

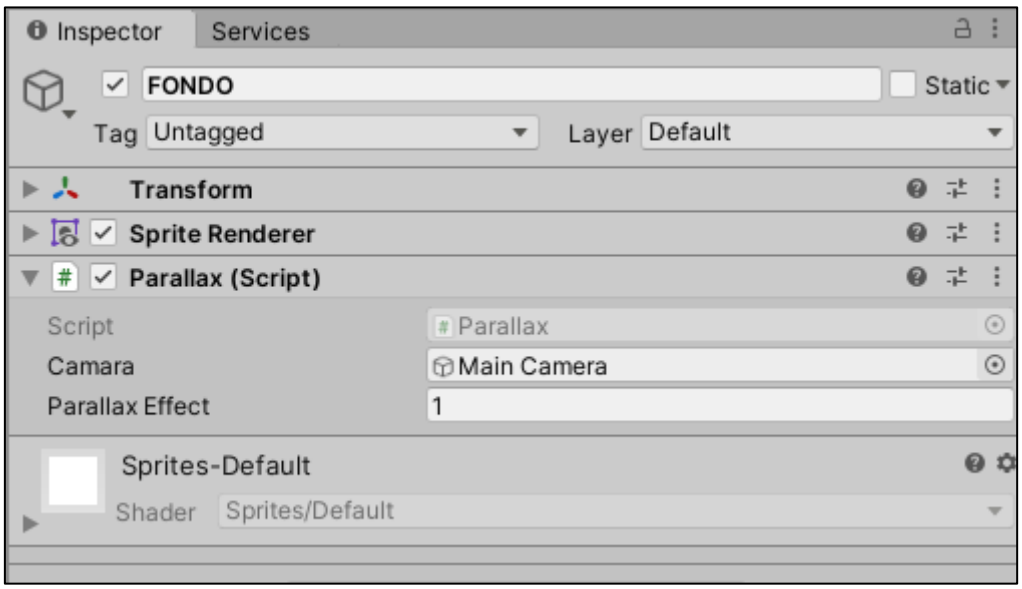

Ilustración 52 Componentes del gameObject Fondo - Fuente: elaboración propia

7.4.2 Escena Principal

Esta escena es la que contiene toda la lógica para poder jugar una partida. Desde el *background* en movimiento del fondo, pasando por la generación y destrucción de mundo, hasta la interacción con los scripts de servicios de terceros.

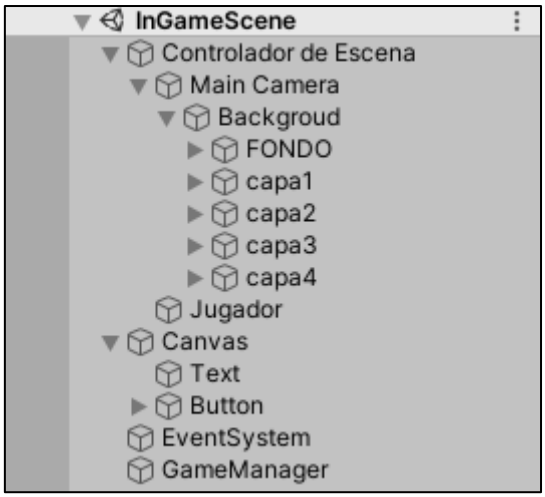

Ilustración 53 GameObjects de la escena principal - Fuente: elaboración propia

Como se puede observar en la ilustración 53, mantiene una estructura de *gameObjects* muy parecida a la escena menú principal. Y no solo la estructura, el efecto *parallax* se hace exactamente igual que en el punto anterior, cada capa del fondo tiene un componente con el script auxiliar Parallax, y este le da la sensación de movimiento.

Sin embargo, aunque la estructura sea semejante, hay grandes diferencias en los componentes que poseen estos *gameObjects*.

La primera de ellas se encuentra en los componentes del *gameObject* **Controlador de Escena**:

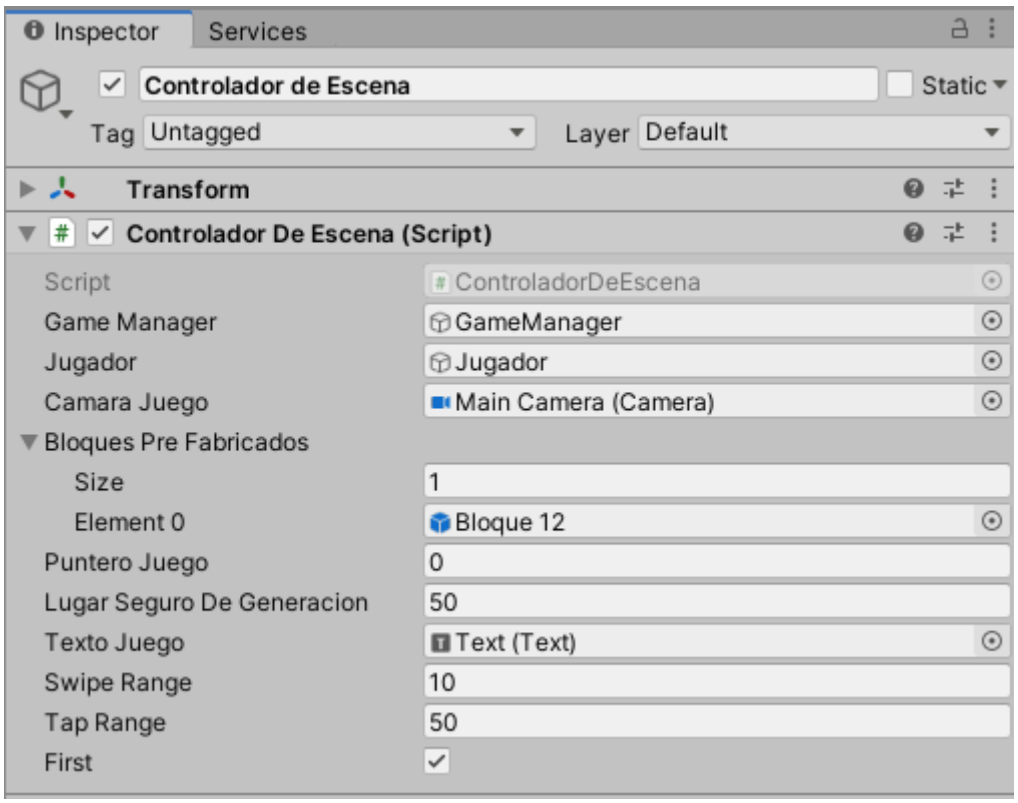

Ilustración 54 GameObject Controlador de Escena - Fuente: elaboración propia

Tiene asociado un script propio, encargado de crear y destruir mundo, mostrar la puntuación en partida, mover la cámara y alguna funcionalidad auxiliar. Hay más información en el siguiente punto.

El *gameObject* Controlador de Escena, también contiene al personaje principal y toda su lógica. Como este objeto es bastante complejo, tiene un punto propio donde se habla sobre su implementación. Se puede ver aquí.

Como apreciamos en la ilustración 53, hay otro Canvas encargado de mostrar la puntuación. Esta puntuación también se gestiona desde esta escena. Pero al igual que con el personaje, tiene empaque suficiente como para tener apartado propio. Ver aquí.

Para terminar este punto, se puede percibir que en las dos escenas vistas previamente, hay un objeto llamado GameManager. Esto es un objeto auxiliar, el cual posee una serie de métodos reutilizables por otras escenas. Son métodos asistentes. En la escena anterior no se ha mencionado porque no tiene prácticamente nada que mostrar, pero sigue la misma estructura que en esta escena. Un *gameObject* con una serie de scripts que sirven para no recargar la lógica básica de otros scripts más complejos.

La escena principal contiene los siguientes componentes:

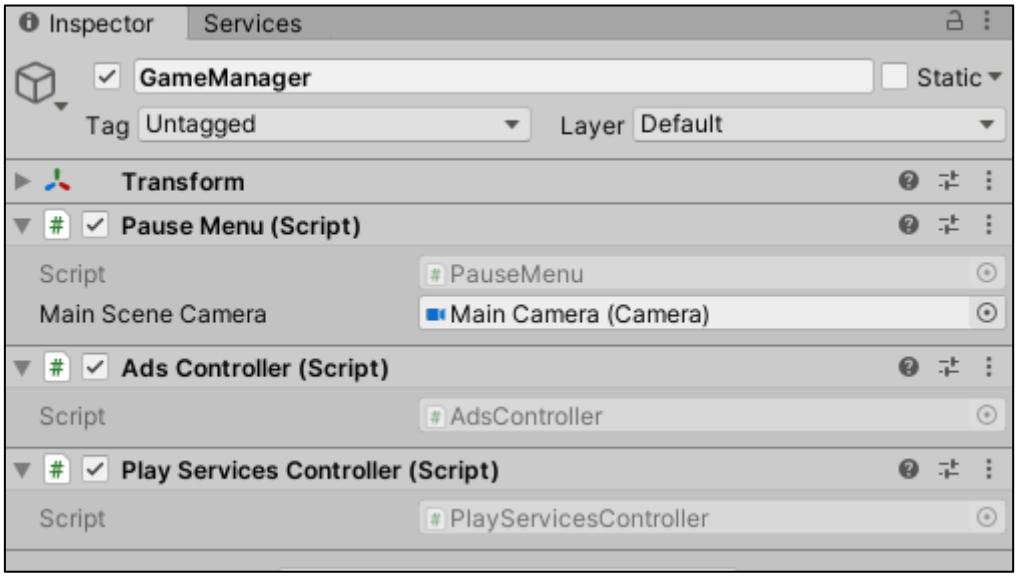

Ilustración 55 Componentes del GameManager - Fuente: elaboración propia

Se pueden ver tres scripts distintos, uno con la lógica del menú de pausa y la cámara principal. Este es el encargado de cargar la escena de pausa y, lleva asociada la cámara principal de juego, para poner en la misma posición la cámara del menú de pausa. Otro, que es el script encargado de mostrar los anuncios, y por último el script que controla los servicios de Google.

7.4.3 Escena Menú Pausa

Esta escena es más sencilla que las dos anteriores. Presenta los siguientes *gameObjects*:

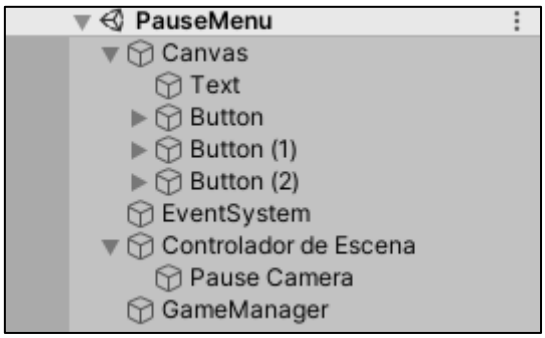

Ilustración 56 GameObjects escena de pausa - Fuente: elaboración propia

Al igual que con las dos escenas anteriores y, principalmente con la escena menú principal, tiene gran cantidad de similitudes con la estructura de objetos.

La principal diferencia, es que no tiene ningún elemento que cubra la pantalla entera. Los únicos objetos visibles desde la pantalla de juego son los botones. Por lo tanto, cuando estamos en el menú de pausa, vemos la partida que tenemos pausada de fondo, más los botones del menú de pausa. Esto se puede apreciar en la ilustración 43.

El resto de los objetos no contiene ninguna lógica compleja. El *gameObject* GameManager, sirve exactamente para lo mismo que los vistos con anterioridad. Contiene un componente con un script auxiliar, encargado de gestionar el cambio entre escenas.

7.4.4 Escena Controles

Esta es la escena con menos carga y no hay puntos especiales que destacar. Como se puede ver en la ilustración 45, presenta una captura con los controles y un botón para volver. Por lo tanto, la estructura es muy similar a la de la escena de pausa vista en la ilustración 56.

7.5 Personaje principal

Como hemos visto en el apartado escena principal, la lógica responsable de gestionar al personaje principal se encuentra contenida dentro del *gameObject* Jugador.

Este objeto llamado Jugador tiene una gran cantidad de componentes distintos, tanto scripts propios como algunos componentes nativos de Unity.

Presenta los siguientes componentes:

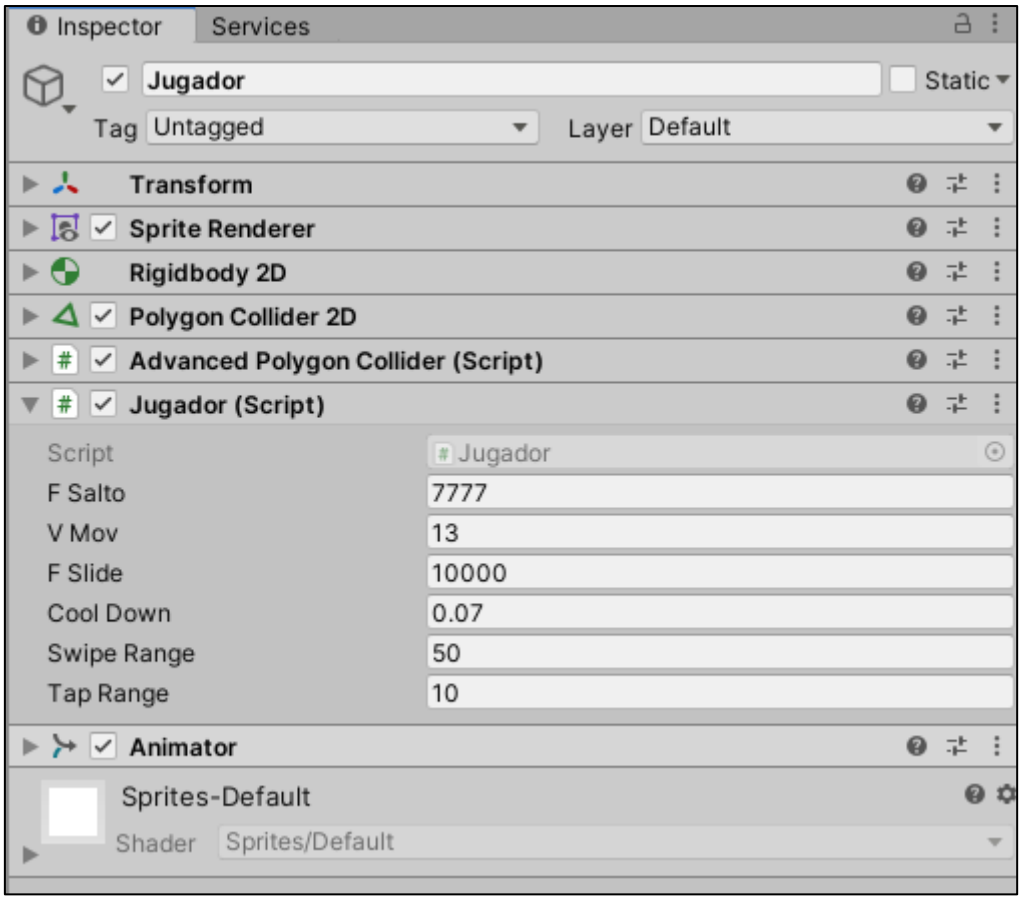

Ilustración 57 Componentes de gameObject Jugador - Fuente: elaboración propia

Notamos una serie de componentes nativos de Unity, los cuales podemos dividir en dos bloques: relacionados con la *hitbox* y relacionados con la animación.

Relacionados con la *hitbox*

- Rigidbody 2D (propio de Unity): pone al objeto al servicio del motor de físicas. Por lo tanto, otorga físicas a un objeto.
- Polygon Collider 2D (propio de Unity): proporciona a un objeto una especie de envoltorio o caja que rodea al objeto en cuestión y es capaz de colisionar con el resto de los objetos.
- Advanced Polygon Collider (no desarrollado por Unity): interacciona con el componente *Polygon Collider* 2D y, va generando una nueva caja de colisiones cada vez que camba el *Sprite Sheet*. Además, debido a que el coste computacional de esta acción es considerable, es capaz de generar una cache para que solo se calcule en la primera iteración. Todo esto es muy útil, para tener un objeto animado con un *Sprite Sheet* que a su vez tiene un *Polygon Collider* dinámico.

Con la unión de estos tres componentes, se consigue un objeto con una caja de colisiones dinámica, apto para detectar colisiones con otros objetos.

Relacionados con la animación

- Sprite Renderer: renderiza el *Sprite* y controla como se ve visualmente en la escena activa.
- Animator: componente capaz de renderizar un *Sprite Sheet*. Genera una serie de eventos controlables por interfaz que proporcionan bastante versatilidad.

Con estas dos herramientas, Unity genera una interfaz parametrizable que se puede gestionar mediante la interfaz del propio motor o con algunos trozos de código. En este caso, se han utilizado ambas opciones.

Definiendo una serie de variables a través de la interfaz de Unity y haciendo el correcto flujo entre transiciones, se puede obtener algo tan simple y visual como el siguiente diagrama:

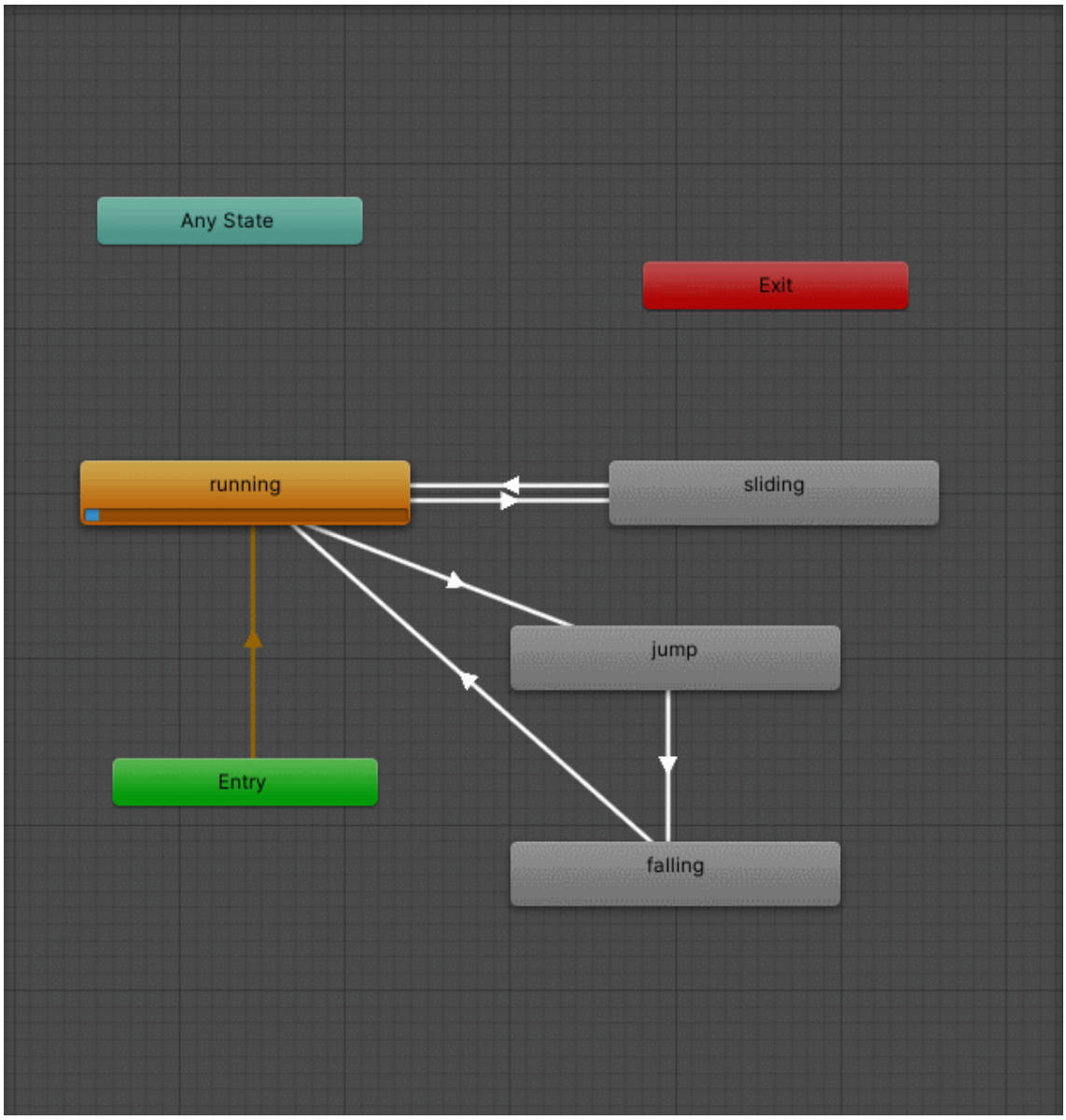

Ilustración 58 Diagrama de flujo de animaciones - Fuente: elaboración propia

Sumado a esto, se realizan algunas comprobaciones en el script auxiliar Jugador, que interfiere también en el flujo natural entre las distintas animaciones.

7.6 Generación y destrucción de mundo

La generación de mundo y destrucción del mismo es algo muy importante en un juego en el que las partidas son de duración infinita, debido a que si todos los objetos que se van creando no se fueran destruyendo, la partida cada vez ocupara más memoria RAM en el dispositivo en el que se esté jugando.

Respecto a la generación de mundo, no se ha elegido un esquema procedural. Es decir, no todo se genera de forma aleatoria. Hay unos bloques predefinidos que se van a asociando a un conjunto de elementos conocido. Esta asociación se hace desde la interfaz de Unity, en un componente que tiene asociado el objeto Controlador de Escena dentro de la escena principal.

Estos bloques predefinidos están formados por un conjunto de *gameObjects* que suelen ser: escenario por donde corre el personaje y enemigos con los que puede encontrarse. Suelen tener la siguiente estructura de componentes:

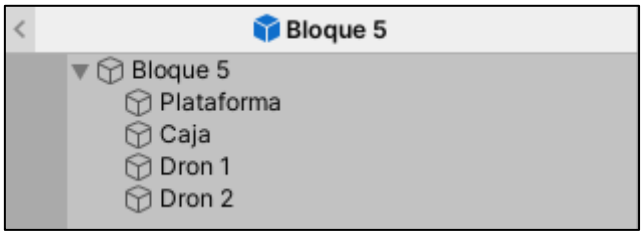

Ilustración 59 Estructura de un bloque - Fuente: elaboración propia

Volviendo al manejo de mundo, hablemos de la destrucción de este. Se hace una vez el personaje haya pasado dos bloques más allá de lo que nosotros vemos por pantalla y, al estar hecho por bloques, es muy simple porque supone eliminar el bloque entero.

Para más información sobre el código con el que se consigue esto, se podrá ver en apartados posteriores.

7.7 Enemigos

Se presentan en la escena como *gameObjects* contenidos dentro de un bloque. Todos los enemigos tienen componentes muy similares, eso sí, cada uno tiene su propia caja de colisiones, su propio Sprite, etc.

La estructura básica de componentes de un enemigo es la siguiente:

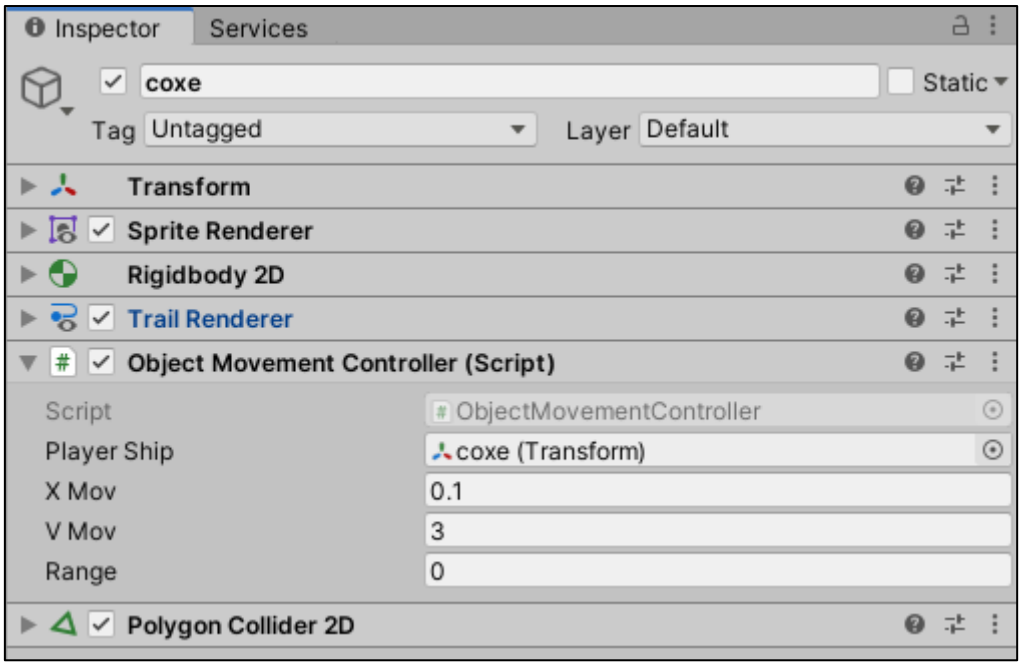

Ilustración 60 Componentes de un Enemigo Coche - Fuente: elaboración propia

Según apreciamos, tiene algo similar al personaje principal. Tiene una serie de componentes que ya hemos visto previamente, los cuales capacitan a cada uno de los enemigos como objetos hábiles para interactuar con el resto de *gameObjects*.

Por otro lado, hay algunos enemigos como el de la ilustración 60, que tiene dos componentes que no todos los enemigos comparten. Estos son el Trail Renderer y el Object Movement Controller.

El Trail Renderer es un componente nativo de Unity capaz de dotar de una especie de estela a los objetos en movimiento. Esto se ha utilizado como un objeto de diseño y se puede ver en la ilustración 35.

Un ejemplo de configuración para uno de los enemigos del proyecto podría ser el siguiente:

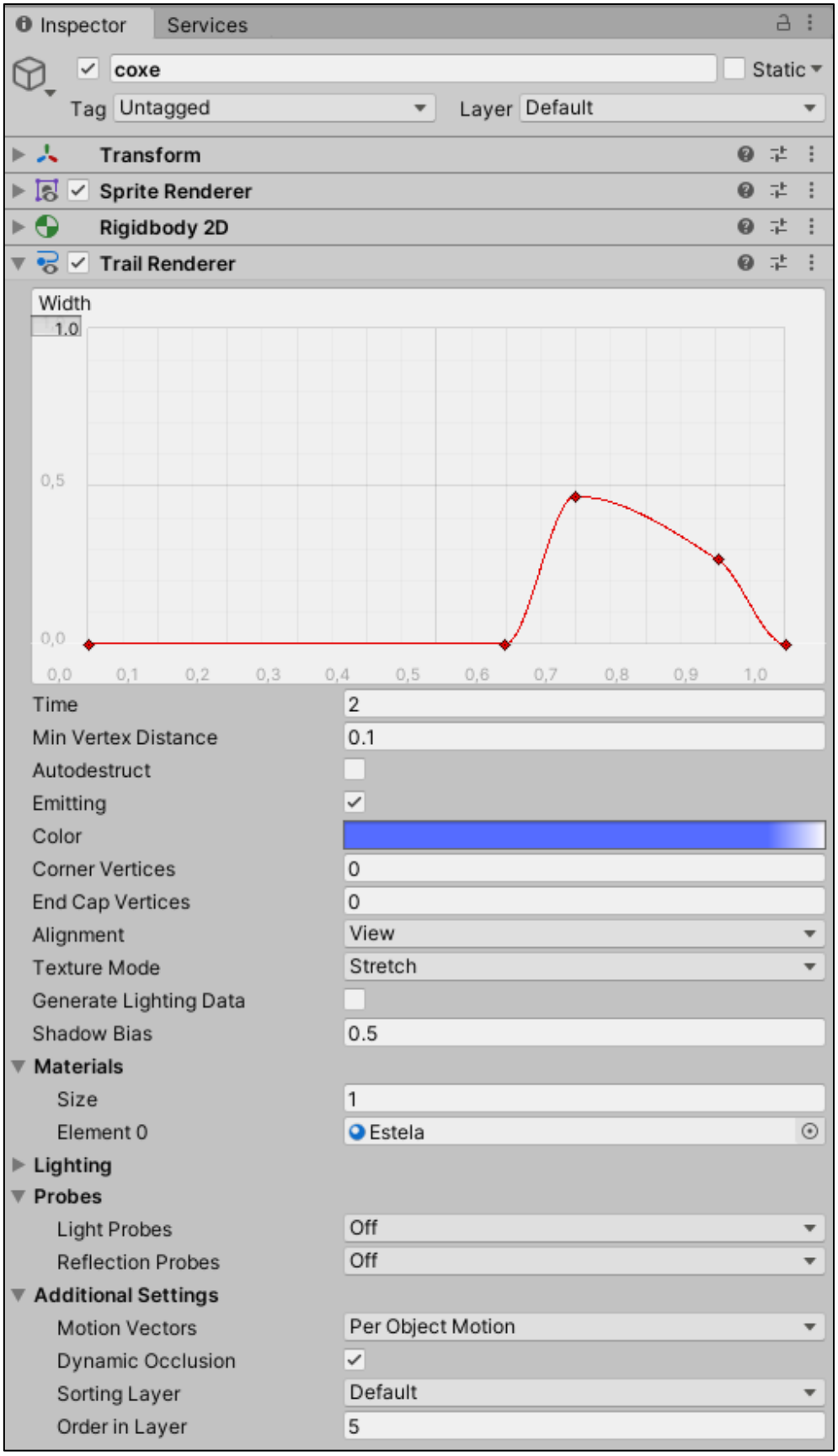

Ilustración 61 Ejemplo configuración de Trail Renderer - Fuente: elaboración propia

Prácticamente todo es personalizable. Se puede ver cómo se puede configurar el ancho de la estela a lo largo del tiempo, el color, el material e incluso algunas cosas más técnicas como la gráfica de la estela en función del tiempo y el ancho.

Por otro lado, el Object Movement Controller es un script auxiliar que se encarga de darles velocidad a los objetos y por tanto de que estén en movimiento. Es un script sencillo que se explica más adelante. Puede verse aquí.

7.8 Estadísticas y logros

El juego tiene estadísticas gracias a las herramientas que provee Google, en concreto gracias al Google Play Console. Este servicio, aparte de para poder publicar y llevar un control de versiones de tu proyecto, también sirve para poder tener en la nube una serie de datos importantes para tu proyecto.

A través de estas herramientas proporcionadas por Google y fácilmente personalizables, se consigue introducir estadísticas y logros por usuario, sin la necesidad de tener nosotros el control sobre ellas. No es necesario un servidor propio para almacenar base de datos, ni el control ni gestión de estos.

Gracias a la personalización que ofrece, podemos crear logros en función de eventos que puedan ocurrir en el transcurso de una partida al juego. Asimismo, nos permite definir como queremos la tabla de marcadores del juego, ya sea mediante la elección de la unidad a mostrar, el orden de la misma e incluso iconos a mostrar.

Sin embargo, aunque Google gestione todo esto, necesitamos un script propio que tenga las credenciales necesarias y los métodos auxiliares requeridos para poder interactuar con los servidores de Google y, por tanto, con su servicio de Google Play Console. Puede verse aquí.

7.9 Scripts

En este apartado veremos la parte más técnica de la memoria. Se mostrará directamente fragmentos de código con los que se ha conseguido implementar el juego y desarrollar sus funcionalidades.

Todos los puntos que se van a mostrar a continuación son scripts que se introducen como componentes dentro de uno o varios *gameObjects* y, le aportan la funcionalidad extra que Unity no tiene de forma nativa.

Es importante introducir de forma previa unos conceptos sobre los scripts que genera Unity.

Veamos una serie de características base y los métodos que trae por defecto:

- Las variables públicas se pueden asignar desde la interfaz de Unity. Las privadas solo desde código.
- Método Start: se llama antes del render del primer fotograma. Sirve como constructor para inicializar variables.
- Método Update: se llama una vez por fotograma. Se utiliza para algunas comprobaciones y, para la asignación de valores.
- Método FixedUpdate: misma funcionalidad que update pero dependiendo de la configuración de pulsos del motor de físicas, se puede ejecutar ninguna, una o incluso varias veces por fotograma (Unity, 2021).

Una vez explicada la lógica básica de los scripts de Unity, vayamos a los propios del proyecto.

7.9.1 Controlador de escena

Es el script auxiliar que utiliza la escena principal para controlar la cámara y, principalmente, la generación de mundo.

El método update de este script, se encarga del sistema para mostrar la puntuación, de la utilización del script de entrada táctil y también de un método que es el encargado de generar y destruir mundo:

```
    private void GenerateAndDestroyWorld()
    {
        if (Jugador != null && PunteroJuego < Jugador.transform.position.x +
LugarSeguroDeGeneracion)
        {
            int indiceBloque = Random.Range(0, BloquesPreFabricados.Length - 1);

            //Debug.Log(BloquesPreFabricados.Length - 1);

            if (PunteroJuego < 0)
            {
                indiceBloque = BloquesPreFabricados.Length - 1; // ESTO ES PARA
PODER ELEGIR EL PRIMER BLOQUE A GENERAR
            }

            GameObject ObjetoBloque =
Instantiate(BloquesPreFabricados[indiceBloque]); // PARA INSTANCIAR UN BLOQUE
NUEVO CON EL INDICE ALEATORIO GENERADO
            ObjetoBloque.transform.SetParent(this.transform); // ASOCIAMOS EL
NUEVO BLOQUE CREADO AL CONTROLADOR DE ESCENA


            Bloque bloque = ObjetoBloque.GetComponent<Bloque>(); // COGEMOS EL
BLOQUE DE ARRIBA PARA PODER MOVERLO DESPUES
            ObjetoBloque.transform.position = new Vector2(PunteroJuego +
bloque.tamanyo / 2, 0); // PARA PONER EL SIGUIENTE BLOQUE DONDE CORRESPONDE
```

```
            if (transform.childCount == 6)
                Destroy(transform.GetChild(2).gameObject); // PARA DESTRUIR LOS
BLOQUES ANTERIORES


            PunteroJuego += bloque.tamanyo;
        }
    }
```

Por último, tiene también un pequeño método llamado `ManageExternalServices()` que es el responsable de conectar con el controlador de anuncios y con el controlador de servicios de Google Play.

7.9.2 Jugador

Al igual que el anterior, es un script que se utilizará para el cálculo de algunos datos de un gameObject, en este caso el del jugador. Por ello necesitará tener un método update() donde reasigne algún valor. }

De su update, podemos destacar que al igual que el anterior, usa el controlador de entrada táctil para saber que acciones está realizando el usuario sobre el dispositivo y que, en función de lo que este le devuelva, modifica el flujo de animaciones visto en el punto de personaje principal. También se puede apreciar como modifica algunos valores del RigidBody2D, recordemos que este es el componente que le proporciona físicas al objeto:

```
    private void FixedUpdate()
    {
        SetAnimationState();

        if (jumping == true)
        {
            anim.SetBool("isJumping", true);

            this.GetComponent<Rigidbody2D>().velocity = (new Vector2(vMov, 0));
            this.GetComponent<Rigidbody2D>().AddForce(new Vector2(0, fSalto));

            jumping = false;
        }
    }
```

Como vemos, modifica la velocidad y le añade la fuerza de salto desde aquí. Esto es muy importante por lo explicado en la definición de update y de fixedUpdate. No se actualizan el mismo número de veces y si, modificas algunos valores del RigidBody desde el update, puedes llegar a generar fallos en el juego.

La única lógica distinto al resto está almacenada en este método:

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        tocandoSuelo = true;
        if (collision.collider.gameObject.CompareTag("Obstaculo"))
            GameObject.Destroy(this.gameObject);

    }
```

Esto, sirve para saber si el usuario ha chocado con algún enemigo y, por lo tanto, la partida debe acabar. Se consigue mediante las etiquetas personalizables que proporciona Unity en sus *gameObjects* y gracias a las cajas de colisiones que ya hemos visto, los *boxCollider*.

7.9.3 Parallax

Sirve para conseguir que el fondo, aunque sean siempre las mismas imágenes, se vayan cambiando de posición del final al principio de forma infinita. Es un efecto visual que ya hemos visto aquí y que, se consigue mediante los siguientes métodos:

```
    void Start()
    {
        startPos = transform.position.x;
        length = GetComponent<SpriteRenderer>().bounds.size.x;
    }
```

En el método start (recordemos que es un método nativo de Unity que se ejecuta una sola vez antes de que el primer fotograma se renderice), cogemos la posición inicial para tener un punto de referencia sobre el cual ir calculando luego y, además como cada *gameObject* o conjunto de *gameObject* tiene su propio script Parallax asociado, necesitamos tener la longitud del Sprite para poder ir moviéndolo del final al principio.

Una vez se ejecuta el start, tenemos un fixedUpdate con lo siguiente:

```
    void FixedUpdate()
    {
        float temp = (camara.transform.position.x * (1 - parallaxEffect));
        float dist = (camara.transform.position.x * parallaxEffect);

        transform.position = new Vector3(startPos + dist, transform.position.y,
transform.position.z);

        float x = -1;
        float test = (float)0.6;
```

```
        if (parallaxEffect == test)
            x = (float)0.5;

        if (temp > startPos + length -10) startPos += length + x;

    }
```

Es muy simple, con las dos variables recogidas en el start y, teniendo como referencia la posición de la cámara principal, va moviendo de la parte de la izquierda de la escena a la derecha, sin la necesidad de tener que crear nuevos *Sprites* ni de tener que estar borrando los que hay.

7.9.4 Controlador de entrada táctil

El controlador de entrada táctil consta de un método que, a partir de las herramientas y objetos que proporciona Unity, usa dos variables públicas para definir que distancias se consideran como un deslizamiento en la pantalla y que distancias se consideran un pulso sobre la pantalla.

Con toda esta información, el método devuelve la cadena de lo que ocurre en pantalla y, vacío en caso de que no haya desplazamiento o pulso. Se consigue mediante el siguiente método:

```
    public string touchInput()
    {

        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Moved)
        {
            currentPosition = Input.GetTouch(0).position;
            Vector2 Distance = currentPosition - startTouchPosition;

            if (!stopTouch)
            {

                if (Distance.x < -swipeRange)
                {
                    stopTouch = true;
                    return "left";
                }
                else if (Distance.x > swipeRange)
                {
                    stopTouch = true;
                    return "right";
                }
                else if (Distance.y > swipeRange)
                {
                    stopTouch = true;
                    return "up";
                }
```

```
                else if (Distance.y < -swipeRange)
                {
                    stopTouch = true;
                    return "down";
                }

            }

        }

        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Ended)
        {
            stopTouch = false;

            endTouchPosition = Input.GetTouch(0).position;

            Vector2 Distance = endTouchPosition - startTouchPosition;

            if (Mathf.Abs(Distance.x) < tapRange && Mathf.Abs(Distance.y) <
tapRange)
            {
                return "tap";
            }

        }

        return "";
    }
```

7.9.5 Controlador de velocidad de enemigos

Este script no posee gran cantidad de lógica. Tiene tres variables públicas que se podrán inicializar desde la interfaz del componente en Unity.

Con estas tres variables, se hacen algunos cálculos para que los enemigos vayan más rápido o más despacio, en función de los valores que se introduzcan desde Unity.

7.9.6 Controlador de anuncios

El controlador consiste en dos variables privadas que almacena las credenciales que Unity genera para tu proyecto, en la inicialización del servicio encargado de proveerte de estos durante la ejecución de tu aplicación y en un método auxiliar capaz de mostrar por pantalla el anuncio.

Conseguimos una integración sencilla gracias a la utilización de Unity Ads, herramienta propia de Unity que es francamente rápida de usar.

Como aspecto curioso, estos son los métodos con los que inicializamos el servicio a través del start y, luego llamamos a este para que nos devuelva un anuncio.

```
    void Start()
    {
        // Initialize the Ads service:
        Advertisement.Initialize(gameId, testMode);
    }

    public void ShowInterstitialAd()
    {
        // Check if UnityAds ready before calling Show method:
        if (Advertisement.IsReady())
        {
            Advertisement.Show();
        }
        else
        {
            Debug.Log("Interstitial ad not ready at the moment! Please try again
later!");
        }
    }
```

7.9.7 Controlador de servicios Google Play

Sin embargo, este controlador es algo más complejo. Necesitamos tener una variable privada con el ID de la tabla de marcadores creada con la Google Play Console. Asimismo, es imprescindible tener una lista de valores con los ID de cada uno de los logros que el usuario pueda conseguir en el transcurso de una partida.

Con esta información obtenida, luego es bastante similar a lo visto en el script anterior. Inicializamos un servicio con la configuración que tenemos en nuestro proyecto compilado y, intentamos hacer un login a los servicios de Play Games con las credenciales del usuario. Estos se obtienen del dispositivo móvil con el que se juegue.

```
    void Start()
    {
        try
        {
            PlayGamesClientConfiguration config = new
PlayGamesClientConfiguration.Builder().Build();
            PlayGamesPlatform.InitializeInstance(config);
            PlayGamesPlatform.DebugLogEnabled = true;
            PlayGamesPlatform.Activate();
```

```
            Social.localUser.Authenticate((bool success) => { });
        }
        catch (Exception exception)
        {
            Debug.Log(exception);
        }
    }
```

Una vez levantado el servicio y con el login del usuario hecho, se dispone de una serie de métodos que nos conectan con el servicio levantado, a los cuales les vamos pasando los ID definidos en las variables privadas, y que acabaran actualizando la base de datos de Google que hemos definido previamente en Google Play Console (herramienta para desarrolladores de Google).

```
    public void AddScoreToLeaderboard(int playerScore)
    {
        if (Social.localUser.authenticated)
        {
            Social.ReportScore(playerScore, leaderboardID, success => { });
        }
    }

    public void ShowLeaderboard()
    {
        if (Social.localUser.authenticated)
        {
            Social.ShowLeaderboardUI();
        }
    }

    public void ShowAchievements()
    {
        if (Social.localUser.authenticated)
        {
            Social.ShowAchievementsUI();
        }
    }

    public void UnlockAchievement()
    {
        if (Social.localUser.authenticated)
        {
            Social.ReportProgress(achievementID, 100f, success => { });
        }
    }
}
```

7.10 Herramientas de desarrollo

En este punto vamos a ver las herramientas que han sido utilizadas para conseguir llevar a cabo el desarrollo del proyecto. Además de Unity y Unity Ads las cuales ya hemos visto y explicado, se han usado dos más que resultaban imprescindibles para llevar a cabo el proyecto, y que se explican a continuación.

7.10.1 Visual Studio

Visual Studio es un conjunto de herramientas desarrollo para desarrolladores. Es, por tanto, un programa que se puede usar para crear programas y aplicaciones (Microsft, 2021).

Visual Studio tiene compatibilidad directa con Unity, hasta el punto de que permite depurar el código en tiempo real. Es decir, te permite ir línea por línea del código viendo en tiempo real que está ocurriendo.

Debido a esto, se ha utilizado Visual Studio para la edición de todos los scripts creados en Unity.

7.10.2 Google Play Console

La **consola de Google Play** es el nexo de unión entre desarrolladores y usuarios. Por ella pasan todas las aplicaciones que se publican para más de dos mil millones de dispositivos activos a lo largo de todo el mundo (Rodríguez, 2019).

Además de ser necesario para el proceso de publicación de cualquier aplicación para la Play Store, ofrece multitud de herramientas que ayudan mucho en el desarrollo de una aplicación. Entre estas herramientas podemos destacar el control de versiones, los procesos de prueba de aplicaciones que presenta y la facilidad para crear marcadores o logros para una aplicación.

Todos esos instrumentos han sido empleados para poder llevar a cabo y, llegar a poder publicar el proyecto en la tienda de Google.

8 Resultados

Como se podrá ver en el apartado de Conclusiones, el objetivo principal de desarrollar y publicar un juego para la tienda Play Store de Google ha sido finalizado con éxito.

Para poder publicar el videojuego en esta plataforma ha sido necesario realizar el siguiente proceso:

1. En primer lugar, habrá que darse de alta como desarrollador de Google. Para ello, se deberá rellenar un formulario a través del siguiente enlace.
2. Una vez el formulario haya sido completado, habrá que pagar la tasa correspondiente para darse de alta. Esta es de 25 dólares, unos 20 euros aproximadamente.
3. Después, una vez recibida la notificación de confirmación por parte de Google, habrá que configurar el proyecto en la plataforma Play Console.
4. Con el proyecto dado de alta y configurado, será necesario compilar la aplicación que se desea subir.
5. Para configurar Unity correctamente, se deberá rellenar las opciones de compilación donde se tendrá que introducir datos como: el número de versión de compilado, el nombre del proyecto, el logo o incluso la organización encargada de desarrollar la aplicación.
6. Cuando las opciones estén correctamente configuradas, habrá que generar el archivo compilado. Unity facilita el trabajo y quitando algún caso excepcional, con compilar el proyecto en el entorno requerido será suficiente.
7. Cuando la aplicación esté compilada y haya un archivo que contenga toda la información de esta, se deberá subir al proyecto que hemos configurado en el punto 3.
8. Por último, habrá que esperar a Google valide el archivo subido y lo dé de alta en la Play Store.

Ahora bien, una vez se ha publicado a través de la herramienta Play Console, se podrá visualizar una gran cantidad de datos útiles capaces de aportar información sobre la monetización, la experiencia de uso e incluso cosas tan necesarias como el número de descargas o impresiones.

Esta información es útil si se le desea dar continuidad al proyecto desarrollado. En este caso, puesto que se desea proseguir con el desarrollo del videojuego más allá de lo descrito a lo largo del documento, se mantendrá un seguimiento constante a toda la información que esta plataforma proporcione y, además, se proveerá de una serie de vías tanto internas como externas a Google, para que cualquier persona que juegue al videojuego y encuentre un error o algún apartado que le disguste, pueda notificar al desarrollador.

Todos estos datos habrá que revisarlos periódicamente para comprobar si las impresiones previstas se están cumpliendo o si, los requerimientos previstos están funcionando correctamente. Además, gracias a otros datos como el número de errores de ejecución, podríamos chequear si los requisitos no funcionales relacionados con la estabilidad del sistema se están cumpliendo.

Con el proyecto desarrollado y publicado, solo falta esperar el tiempo suficiente para ver cuánta atención recoge.

9 Conclusiones

Una vez finalizado el proyecto, es posible extraer las conclusiones y evaluar el proyecto con los resultados obtenidos. Estas conclusiones han sido divididas en Conclusiones de los objetivos y Conclusiones sobre el diseño y desarrollo.

9.1 Conclusiones de los objetivos marcados

En primer lugar, cabe señalar que todos los objetivos propuestos originalmente han sido conseguidos, aunque algunos de ellos han sufrido pequeñas variaciones. Aun así, se puede decir que se ha logrado el objetivo principal de proyecto, desarrollar el videojuego y sus distintos apartados.

Pero desarrollar el videojuego tiene multitud de subapartados, así que a continuación se van a exponer los distintos objetivos propuestos inicialmente y el resultado obtenido:

- **Aprendizaje del motor Unity, y por tanto del lenguaje de programación C#. Todo el proyecto estará basado en el motor 2D de Unity:** el proyecto ha sido finalizado con éxito, y el juego ha conseguido ser un producto real. Para que todo esto llegue a pasar, primeramente, se han tenido que sentar las bases de Unity para poder avanzar en el proceso de desarrollo del juego. Entonces, podríamos afirmar que este objetivo ha sido logrado con éxito y que, además, se han sentado las bases necesarias para la continuidad de este proyecto, e incluso, para el desarrollo de futuros proyectos basados en el mismo motor.

- **Utilización de las metodologías ágiles vistas a lo largo del grado académico. Todas estas herramientas serán de gran ayuda para la gestión y planificación de todo el proyecto:** uno de los puntos más críticos en el desarrollo de un software es la planificación, gestión y supervisión de este. Sin un correcto sistema encargado de estas tareas, cualquier proyecto podría presentar problemas a lo largo de su desarrollo. En este caso, se han definido una serie de estrategias a seguir, las cuales podemos ver en el apartado metodología. En concreto se han diseñado dos estrategias distintas a seguir: una encargada del desarrollo del proyecto y otra, responsable de la gestión de este. Ambas se han intentado seguir de la forma más fiel posible, donde además ha intervenido la correcta supervisión del tutor del proyecto. Con todo esto, se puede llegar a decir que el objetivo ha sido conseguido.

- **Definición e integración del modelo de negocio. Este negocio, estará basado principalmente en publicidad, que se mostrará de manera estática en algunos apartados del juego, y también en forma de vídeos en otros apartados. Por ejemplo, en el caso de perder la partida, tendrás la oportunidad de seguir si visualizas el vídeo de un anuncio:** desde el comienzo del desarrollo, este apartado fue fijado como otro pilar necesario para considerar el proyecto como finalizado. Hay que decir que ha habido pequeñas variaciones en la elección de los anuncios a mostrar debido a que se ha querido impulsar la experiencia de juego, finalmente se optó por anuncios intersticiales. Toda esta elección y su debida justificación puede verse en el apartado de modelo de negocio. Puesto que se ha conseguido llegar al punto en el que el juego tiene una estrategia de mercado clara y va generando beneficios de forma pasiva, se

puede llegar a afirmar que este objetivo ha sido logrado con éxito con los pequeños cambios mencionados.

- **Diseño e implementación de todos los apartados que englobe el videojuego. Desde un mundo de generación infinita, al diseño de este, pasando por el sonido, el personaje, el fondo visual e incluso por los distintos bloques que se acaban convirtiendo en un escenario que se genera infinitamente:** todos los apartados de diseño y desarrollo han sido cumplidos sin problemas, aunque por momentos ha sido más complicado de lo previsto. El déficit de conocimiento especialmente en áreas de diseño ha sido un lastre que se ha ido arrastrando durante todo el proyecto. No obstante, una vez hubo una base de conocimientos tanto de diseño como de los programas requeridos, el proceso empezó a coger forma con celeridad.

- **Generación del fichero en formato .APK necesario para su publicación en la Play Store de Google e integración en esta misma tienda. No solo bastará con generar la .APK a través de Unity, el juego deberá pasar la validación de la Play Store para poder dar por terminado el proyecto:** este objetivo era uno de los más importantes puesto que la publicación del juego debía de ser uno de los pilares del proyecto. La generación del archivo ejecutable ha sido más o menos sencilla, sin embargo, la subida a la Play Store ha sido un proceso el cual ha requerido un cierto aprendizaje, pero que al final ha sido cumplido sin problema.

9.2 Conclusiones sobre el diseño y desarrollo

Seguidamente, se van a exponer las distintas conclusiones extraídas a lo largo de todo el proceso de diseño y desarrollo del proyecto. Estas observaciones se van a dar como una lista de conceptos aprendidos. Serían los siguientes:

- **Reto tras reto**: el desarrollo del proyecto ha supuesto un constante camino de superación. La titulación cursada (ingeniería multimedia) permite en el cuarto curso elegir especializarse entre dos itinerarios distintos: uno de ellos relacionado con el diseño e implementación de contenido Web y otro con las mismas bases, pero orientado hacia la industria de los videojuegos. En este caso, solo se ha cursado el itinerario Web, por lo que los conocimientos sobre el diseño y desarrollo de videojuegos debían ser básicos y no avanzados. Esto es el motivo de la elección del tipo de proyecto. Desde el principio se ha buscado un reto personal y académico, orientado a un aumento de las capacidades como desarrollador de software. No obstante, a mayor reto, mayor dificultad, y el desconocimiento de gran parte de las áreas requeridas durante el transcurso del proyecto, como podría ser el diseño requerido para el juego, ha llegado a retrasar alguno de los apartados necesarios.
- **Gestión y planificación**: todas las dificultades debidas a problemas externos, como podría ser problemas laborales, han evidenciado aún más la estricta necesidad de una correcta gestión y planificación de cualquier proyecto con un mínimo de complejidad. Hay una necesidad imperiosa de dividir el trabajo en tareas, de intentar asignar una estimación temporal y de, siempre en medida de lo posible, cumplir ambas. Esto es algo que ya se había visto a lo largo del último curso académico, pero que aquí se ha

evidenciado sustancialmente debido a las características del proyecto y al reto que suponían.

- **Supervisión**: otro de los puntos que han resultado de gran beneficio para el desarrollo del proyecto y que finalmente, han llegado a ser prácticamente imprescindibles para la finalización con éxito del mismo, ha sido la supervisión del proyecto, en este caso guiada por el tutor encargado. Sin embargo, aunque esto no sea siempre posible, se debería intentar buscar una o varias personas de confianza que posean la capacidad de evaluar el proceso de desarrollo de un proyecto.
- **La importancia de unas buenas bases**: debido a que ha habido momentos de desconocimiento absoluto sobre como empezar a diseñar o desarrollar alguno de los requerimientos necesarios, ha sido necesario un aprendizaje previo para poder llegar a realizar estos requerimientos. Sin embargo, en algunas ocasiones se ha podido observar cómo se apostaba por la celeridad a la hora de intentar resolver estas situaciones. Esto puede conllevar problemas a la hora de entender el propio funcionamiento del código que se está realizando e impidiendo cualquier mejora de este. Esta apuesta por la celeridad y no por intentar asentar unas bases lógicas y coherentes para futuros desarrollos ha sido un error que se ha aprendido durante el proceso de desarrollo y que se tendrá en cuenta tanto en proyectos venideros como con la continuidad de Roboto Runner.
- **Continuidad y escalabilidad**: una de las situaciones más repetidas en el desarrollo de proyectos software, podría ser la necesidad de ampliar los requerimientos o las funcionalidades del mismo. Al ser una industria en constante crecimiento, hay que pensar siempre en cómo se podría continuar el proyecto incluso una vez acabados los requisitos iniciales. En concreto en este caso, se desea seguir añadiendo mecánicas y mejorando la jugabilidad del videojuego para intentar agradar al máximo número posible de usuarios de la Play Store.

Sin lugar a duda, todos estos conocimientos aprendidos, serán una gran ayuda y servirán de experiencia para la gestión, para la planificación y para el desarrollo de los siguientes proyectos.

10 Bibliografía

3Djuegos. (2021). *3Djuegos.* Obtenido de https://www.3djuegos.com/19985/spider-man-unlimited/

adsbalance. (2021). Obtenido de https://adsbalance.com/2021/01/23/how-much-do-apps-earn-from-advertising/

Alice. (27 de 4 de 2020). *Gridpak.* Obtenido de https://www.gridpak.com/es/adobe-illustrator-opiniones-precios-y-donde-comprar-el-software/

Amazon. (2021). *Amazon.* Obtenido de https://www.amazon.es/POCO-X3-Pro-Smartphone-DotDisplay/dp/B08YJ923NY/ref=sr_1_45?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=telefono+android&qid=1625310129&sr=8-45

Amazon. (2021). *Amazon.* Obtenido de https://www.amazon.es/Lenovo-Tab-P11-Snapdragon-ampliables/dp/B08SCBFG97/ref=sr_1_8?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=tablet&qid=1625310233&sr=8-8

Amplio. (04 de 08 de 2018). Obtenido de https://amplioservices.com/blog/how-implement-price-win-process

Apple. (2021). *Apps Apple.* Obtenido de https://apps.apple.com/es/app/procreate/id425073498

Blázquez, F. L. (2014). *Hablando de manzanas.* Obtenido de https://hablandodemanzanas.com/noticias/el-creador-de-flappy-bird-se-deja-ver-y-concede-una-entrevista-rolling-stones

Carrasco, A. C. (04 de 07 de 2018). *UPM.* Obtenido de https://blogs.upm.es/observatoriogate/2018/07/04/que-es-un-motor-de-videojuegos/

CreativosOnline. (14 de 8 de 2014). *CreativosOnline.* Obtenido de https://www.creativosonline.org/adobe-illustrator-que-es-y-para-que-sirve.html

DEV - Asociación Española de Empresas Productoras. (2020). *DEV.* Obtenido de DEV - Desarrollo Español de Videojuegos: https://dev.org.es/images/stories/docs/libro%20blanco%20del%20desarrollo%20espanol%20de%20videojuegos%202020.pdf

GamePlayDeveloper. (2021). *GamePlayDeveloper.* Obtenido de GamePlayDeveloper: https://www.gameplaydeveloper.com/unity-ads-vs-admob-which-earns-more/

GamerDic. (18 de 07 de 2014). *GamerDic*. Obtenido de GamerDic: http://www.gamerdic.es/termino/endless-runner

Gardey, J. P. (2013). *Definicion.de.* Obtenido de Definicion.de: https://definicion.de/videojuego/

Google. (2021). *Google.* Obtenido de Google: https://play.google.com/console/

google. (2021). *thinkwithgoogle.* Obtenido de thinkwithgoogle: https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/paid-vs-free-ap-user-statistics/

Grimm, T. (1998). *The Human Condition: A Justification for Rapid Prototyping.* Time-Compression Technologies.

Indeed. (2021). Obtenido de https://es.indeed.com/career/programador-junior/salaries

infoautonomos. (2019). Obtenido de https://www.infoautonomos.com/blog/cuanto-cuesta-contratar-un-trabajador/

Llasera, J. P. (20 de 07 de 2020). *imborrable.* Obtenido de https://imborrable.com/blog/que-es-procreate/

Martínez, C. (21 de Julio de 2015). *Enter.co*. Obtenido de Enter.co: https://www.enter.co/cultura-digital/videojuegos/gameloft/cual-fue-el-primer-runner-de-la-historia/#:~:text=El%20primero%20es%20'Speed%20Race,era%20m%C3%A1s%20dif%C3%ADcil%20evitar%20obst%C3%A1culos.

Matulef, J. (24 de 5 de 2016). *Eurogamer.* Obtenido de https://www.eurogamer.net/articles/2016-05-24-pac-man-256-is-coming-to-consoles-and-pc-in-june

Microsft. (10 de 02 de 2021). Obtenido de https://docs.microsoft.com/es-es/visualstudio/get-started/visual-basic/tutorial-console?view=vs-2019

Microsoft News Center. (04 de 06 de 2018). *Microsoft.* Obtenido de https://news.microsoft.com/2018/06/04/microsoft-to-acquire-github-for-7-5-billion/

mobygames. (2021). *mobygames.* Obtenido de https://www.mobygames.com/game/frogger

moddb. (2021). Obtenido de https://www.moddb.com/engines/unity

Museum of the Game. (2021). *arcade-museum.* Obtenido de arcade-museum: https://www.arcade-museum.com/game_detail.php?game_id=7857

Nguyen, L. A. (2014). *Forbes.* Obtenido de https://www.forbes.com/sites/lananhnguyen/2014/02/11/exclusive-flappy-bird-creator-dong-nguyen-says-app-gone-forever-because-it-was-an-addictive-product/?sh=7cb76e4d6476

Parkinson, C. N. (1957). *Parkinson's Law, and Other Studies in Administration.* Houghton Mifflin.

PcComponentes. (2021). *PcComponentes.* Obtenido de https://www.pccomponentes.com/millenium-machine-1-mini-rrx7n-amd-ryzen-5-2600-8gb-1tb-250gb-ssd-gtx1650

Rodríguez, T. (14 de 02 de 2019). *Xataka.* Obtenido de https://www.xatakandroid.com/play-store/tripas-consola-google-play-asi-herramienta-fundamental-para-publicar-administrar-aplicaciones-android

Sommerville, I. (2005). *Software Engineering (7ª ed.).* Addison-Wesley.

Sommerville, R. (1998). *Ingeniería del Software. Un enfoque práctico (4ª ed.).* Mc Graw-Hill.

Stellman, A., & Greene, J. (2005). *Applied Software Project Management.*

Tambur, S. (12 de 8 de 2013). *Estonian World.* Obtenido de https://estonianworld.com/technology/start-up-spotlight-toggl/

The new York Times. (14 de 11 de 2017). *The new York Times.* Obtenido de https://www.nytimes.com/2017/11/14/technology/personaltech/chrome-dinosaur-internet-connection.html

Torvalds, L. (s.f.). *YouTube.* Obtenido de https://www.youtube.com/watch?v=4XpnKHJAok8&t=90s

Trello. (2021). *Trello.* Obtenido de https://trello.com/about

Uniovi. (2015). Obtenido de http://www.pulso.uniovi.es/wiki/index.php/Unity

Unity. (2021). *Unity.* Obtenido de https://docs.unity3d.com/es/530/Manual

Unity. (2021). *Unity Technologies.* Obtenido de Unity Technologies: https://unity3d.com/es/unity/faq#:~:text=Unity%20Pro%20La%20nueva%20y,Unity%20Enterprise%3A%20Para%20organizaciones%20grandes.

Unity. (2021). *UnityDocs.* Obtenido de UnityDocs: https://docs.unity3d.com/es/530/Manual/UnityAds.html

Webster, A. (2014). *The Verge.* Obtenido de https://www.theverge.com/2014/6/4/5776232/temple-run-1-billion-downloads

Wiegers, K. E. (2003). *Software Requirements 2: Practical techniques for gathering and managing requirements throughout the product development cycle.* Redmond: Microsoft Press.